IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 7 July 2023, accepted 8 August 2023, date of publication 28 August 2023, date of current version 5 September 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3309149

RESEARCH ARTICLE

Efficient Dissimilarity Detection in Time Series With Application to Side-Channel Analysis

MINE KERPICCI[1], (Graduate Student Member, IEEE),
MILOS PRVULOVIC[2], (Senior Member, IEEE),
AND ALENKA ZAJIĆ[1], (Senior Member, IEEE)
[1]School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA
[2]School of Computer Science, Georgia Institute of Technology, Atlanta, GA 30332, USA
Corresponding author: Mine Kerpicci (mkerpicci3@gatech.edu)

ABSTRACT This paper proposes a dissimilarity detection algorithm that can find different regions between similar signals. In particular, we address the detection problem of branching statements in program control flow using electromagnetic (EM) side-channels, where we have shown that such statements can be detected in emanating side-channels. Based on the findings, we have proposed a generalized approach for dissimilarity detection that can be efficiently applied to various real-world applications. In the proposed method, symbolic representation of the signals is used for efficient processing, where each signal frame is transformed into a string. The codebook of observed patterns is constructed with the reference signal. Then, the sequence of strings that is obtained from the main signal is compared with the codebook to find the newly observed patterns. Finally, the presented method outputs the samples of dissimilar regions in the signal compared to the reference. In the experiments, various EM side-channel signals are collected from different devices and different control flow examples to show the applicability and efficiency of the proposed method and the results show that dissimilarities can be detected with $>98\%$ accuracy.

INDEX TERMS Dissimilarity detection, electromagnetic side-channel, pattern matching, program control flow, time series representation.

I. INTRODUCTION

The tremendous expansion of the Internet of Things and embedded devices in recent years causes an increasing need to track and monitor their process for security against cyber-attacks and software crashes that can be unrecoverable for the user [1], [2]. To prevent these events, program control flow, a sequence of statements that are executed by the device processor is a very valuable tool. One of essential elements to track in the program execution is decision or branching statement, where the program executes different code blocks based on the evaluation of the defined condition with the program inputs. This conditional code execution is crucial in many programming scenarios such as controlling the program flow and error-handling. Therefore, the detection of selection statements is highly important to track the program behavior in various methods developed against cyber-attacks and performance problems such as program profiling [3], [4], [5], tracking [6], [7] and malware detection [8], [9].

The side-channels, as form of information leakage, are considered as valuable sources for program tracking due to their relatable content with the program activities. In the literature, they are obtained in various forms including power consumption [10], [11], electromagnetic emanations [3], [12] or acoustic emissions [13], [14]. It is shown that such information can be efficiently exploited for various tracking purposes such as profiling and malware detection [7], [15], [16]. Among these side-channel forms, EM emanations provide the most advantageous and powerful framework since

The associate editor coordinating the review of this manuscript and approving it for publication was Diana Gratiela Berbecaru.

they can be collected from a longer range without requiring instrumentation. Recent studies with EM side-channels also show that these signals provide more information, which enables more robust tracking and detection [17], [18].

In this paper, we propose a new detection method for finding branching statements in program control flow by using electromagnetic (EM) side-channels without interrupting the processor operations. EM side-channels, as one of the essential information leakage forms, emanate from the device during program running [19]. Since they are constructed as a result of program activities, they contain valuable and sensitive information on the underlying control flow elements [12], [20], [21]. In our proposed approach, we collect and process the EM side-channels in a non-intrusive way to detect dissimilarities caused by selection statements.

The decision-making of a program is controlled by ''if-else'' statements. An executed program evaluates the pre-defined condition with the inputs at the execution time and follows the corresponding path only if the condition is met. Therefore, program executions with different inputs result in variations in the operation sequences and dissimilarities in the emanating EM side-channels. Moreover, the selection statements are commonly interlaced with ''for'' loops where the program evaluates the selection criterion at each loop iteration and follows the corresponding paths. This constructs an EM signal with repetitive patterns and dissimilarities among the repeating regions. Therefore, we propose a detection method to find such dissimilar regions among multiple signals or in one signal with repetitive patterns.

The detection of dissimilarities that are introduced in the EM side-channels as a result of the branching statements is a challenging problem due to the nature of the program control flow, device processor, and sampling issues. One can observe differences even among the signals collected as a result of the same code evaluation due to noise, environmental changes or internal processes of the device. In addition, similar regions can exist among the signals of different codes due to similar operations or functions called inside control flow. Moreover, there are infinite possibilities and combinations in terms of control flow sequence and its resulting EM side-channels, and hence it is not possible to obtain large amount of labeled training in most of the cases. To address this problem, we propose an approach that can be used even when there is little or no prior information available. For this, we transform the collected EM signals into symbolic representations to capture the signal patterns. We construct a code-book with the patterns of the reference signal so that the proposed method is applicable even when there is no labeled training data. Then, we compare the symbolic representation sequence of the main signal with the constructed references for the detection of dissimilar regions. In our experiments, we use two different devices on which programs of various scenarios are running to collect EM side-channels and apply the proposed approach to these signals to illustrate its performance. Through our results, we show that the proposed method can be efficiently used in various scenarios.

The main contributions of this paper are as follows.

- We introduce a dissimilarity detection approach to find different regions among a number of similar signals without requiring labeled training data.
- We address the detection problem of branching statements in program control flow via side-channels in a non-intrusive way without interfering the program activities.
- We propose to combine time series representation and pattern matching for an efficient dissimilarity detection approach that can be generalized to various time series applications.
- We show that selection statements are observed as dissimilar regions in EM side-channels. In our experiments, we illustrate the performance of our method on side-channel signals of various programs running on different devices where we show the effectiveness of the introduced method on real applications.

The organization of the paper is as follows. In Section II, we discuss the related work in the literature. We define our problem setting in Section III. In Section IV, we describe our methodology where we provide a background information and introduce our approach to dissimilarity detection. We demonstrate the performance of our method with side-channel measurements in Section V and conclude with final remarks in Section VI.

II. RELATED WORK

Program tracking is highly studied in the literature due to its crucial role in various cyber-security applications. The traditional approach to this problem requires program instrumentation that is placed before or during the execution [22], [23]. Even though this approach allows one to obtain information in some settings for program monitoring and tracking, it causes resource overhead and changes the program behavior due to the inserted code. To overcome these issues, zero-overhead systems are proposed where EM side-channels are exploited where the tracking and profiling is performed externally without requiring instrumentation [3], [6].

EM side-channels that emanate from the device while operating contain sensitive and valuable information about the system and running program [12], [24]. Therefore, EM side-channel signals are used as essential sources to gain insight by many applications that are developed to improve system performance and security [25], [26], [27], [28].

A program execution monitoring approach based on EM side-channel signals is proposed in [6] where a set of possible code outcomes is created from the known program code in training phase and detection is performed in prediction phase. Similarly, [29] studies control flow monitoring by using a neural network architecture to process EM side-channels. [30] introduces an approach for the detection of anomalies in program execution where malware and code injections are detected based on the spikes in the EM spectrum. Reference [31] also works on malware detection with side-channels where autoencoder based deep learning method is proposed.

As studied in all these works, EM side-channels can be used effectively to track program behavior since they are the direct results of the running program [12], [32]. In our setting, we also use captured EM side-channel signals to detect selection statements without interfering the device process. Even though the previous studies with EM side-channels perform well in several settings, they either require instrumentation and an access to the program code or large number of labeled training data. Therefore, they are not applicable to our problem where we search for selection statements based on observed data sequences without prior knowledge.

Instead, we have developed a new approach to use EM signals for selection statement detection based on an important observation on the nature of the side-channels. The selection statements result in differences, i.e., dissimilarities, in the emanating EM side-channels of the code evaluations with various inputs or parameters. Therefore, a dissimilarity detection approach can be used to search for dissimilarities in the signal whose detection reveals the selection statements in the control flow.

In our work, we propose a general framework which can be used to detect dissimilarities among two or more signals where their patterns are compared with the reference patterns. Moreover, we study the application of our approach to detect dissimilar regions in a single recording when the observed signal contains repetitive patterns. In this case, we propose to use a portion of the signal such as its one iteration to construct the reference patterns via sliding window. In such scenario, one can directly extract single iteration via visual inspection or automatically with loop detection as in [20].

Dissimilarity between sequences or observations is studied in many applications of various data types including text, image or time series data [33], [34], [35], [36]. In such studies, similarity/dissimilarity analysis methods are mostly developed to target best match retrival, clustering and classification applications. Therefore, they are interested in the comparison of entire sequences to represent their relation with each other over a single metric that measures their similarities/dissimilarities such as distance or correlation. For example, [37] uses k-means algorithm with dynamic-time-warping to classify similar time series observations. Reference [38] studies bag-of-patterns approach that is developed based on bag-of-words algorithm for classification based on structural similarities in time series. Similarly, [39] uses bag-of-features framework for time series classification to match the sequences based on their extracted feature similarities. Such methods perform well on their targeted tasks that require high-level modelling and comparison. However, they are not directly applicable to detect local dissimilarity locations between highly similar sequences because sequences in code are very short and do not have enough samples to establish similarity rate with high confidence.

In our study, we aim to develop a method that can be used to compare the sequence of measurements collected at regular intervals over time, i.e., time series, in terms of their local changes, i.e., variations, efficiently. Time series representation techniques are introduced in the literature to map the time series data into a lower dimension and process efficiently to handle with its challenging characteristics [40], [41]. Among these techniques, piecewise aggregate approximation (PAA) is particularly useful to construct an efficient representation of the observed data to reduce the computational cost during processing [42]. It works by dividing the time series into equal-length segments and computing the mean value of each segment. Symbolic aggregate approximation (SAX) is introduced to use the advantages of PAA to capture the overall signal shape as well as text retrieval techniques and text-based algorithms [43], [44]. It converts the time series into a string, i.e., sequence of discrete symbols, based on a predefined breakpoints. Therefore, it has the ability to capture the patterns and trends of the processed data effectively while being resistant to the environmental effects such as noise, which makes it particularly useful. Hence, we use this representation method with a sliding window so that we can capture local shapes, i.e., patterns, in the observed signals efficiently. Moreover, since this results in a set of strings, we propose to combine this with a pattern matching approach, which is originally introduced to search for a specific pattern or template in a given input [45]. Instead of having one template, we construct a set of templates from the observed patterns via time series representation and use it as a reference for newly observed sequences. Hence, for the first time, we combine time series representation and pattern matching approach to address the challenges of the EM side-channel analysis and detect selection statements efficiently. As a result, we not only show that the selection statements in the control flow can be observed as dissimilar regions in the emanating EM side-channels but we also propose an efficient approach to detect such dissimilarities.

III. PROBLEM DESCRIPTION

Selection statements, which are also known as branching statements, are used to decide on the execution path of a program based on certain conditions. These statements are highly used by programmers for several purposes such as code efficiency, input/output validation or recursive calculations. Therefore, most of the program control flows contain selection statements in various forms, and their detection is highly essential to understand the code behavior. Side-channel analysis is a powerful tool to particularly detect such behaviors since side-channels are generated directly through the program activities and they can be collected without interfering the process.

We define the problem of detecting selection statements as a dissimilarity detection problem. The reason is that such statements lead the program to perform different activities based on the evaluation of the branching condition. Hence, they result in dissimilar regions in the branching locations of the resulting side-channel signals. In our work, we provide an approach to detect such dissimilar regions via EM side-channels.

We consider an EM side-channel signal generated as a result of a program code that contains a selection statement in the following form

```
if (condition(i))
    statement₁(i);
else
    statement₂(i);
```

where the program executes `statement`$_1$ if the input `i` satisfies the pre-defined condition. Otherwise, it follows the execution path of `statement`$_2$. In such a code example, the evaluation of the program with different inputs results in different orderings of the executed paths and hence different time series data (side-channel signals). For example, for an input `i`, an emanating side-channel signal is collected as $\boldsymbol{x} = [x_1, x_2, \ldots, x_{N_1}]$. And, for an input `j`, the same program results in a different signal $\hat{\boldsymbol{x}} = [\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_{N_2}]$. In the case of one selection statement, first k samples and last m samples of these signals would be the same with variations in the middle such that $\boldsymbol{x}_{1:k} = \hat{\boldsymbol{x}}_{1:k}$ and $\boldsymbol{x}_{N_1-m:N_1} = \hat{\boldsymbol{x}}_{N_2-m:N_2}$. Then, the problem becomes detecting the dissimilar regions of $\boldsymbol{x}_{k:N_1-m}$ and $\hat{\boldsymbol{x}}_{k:N_2-m}$.

Note that the partial code given above is a simple representation of the branching. In an original code written for real applications, such statements are observed in more complex combinations in different forms. One of their common use is where the selection statements are called inside loops as in the following expression

```
for (int j = 0; j < T; ++j){
    if (condition(i, j))
        statement₁(i, j);
    else
        statement₂(i, j);
}
```

Here, `for` statement that iterates the variable `j` results in execution of the inner statements T times in a loop. At each iteration, the condition of `if-else` is evaluated and either one of the statement paths is executed based on the input `i` and/or iteration number `j`. Ideally, in a `for` loop that performs same operations at each iteration, we observe a single pattern repeating T times in the collected signals. When `if-else` statement is placed inside the `for` loop, this result in two (or more) patterns that are called T times in total due to different operations performed in different paths.

We provide an example of EM side-channel signal collected from device running a program similar to given in the description above in Fig. 1. This signal contains

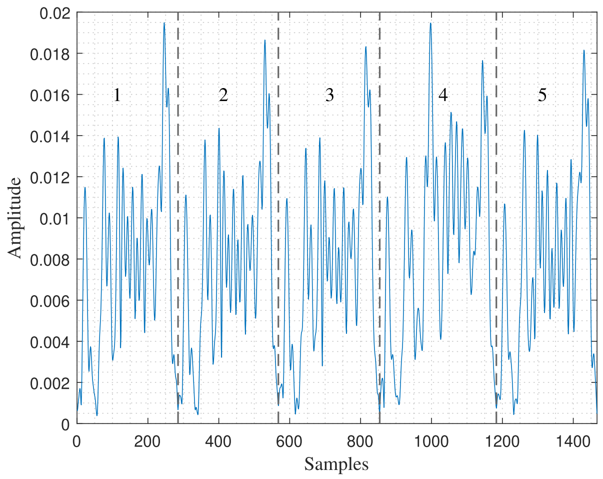


FIGURE 1. **An example of a side-channel signal collected from a device running a program of a loop with 5 iterations and selection statements.**

5 iterations of the executed `for` loop where the 4th iteration performs `statement`$_1$ and the remaining iterations perform `statement`$_2$. Here, we make two observations: (1) Even though the 1st, 2nd, 3rd and 5th iterations are the results of the same statements, they are in different lengths due to different inputs, noise, interference and architecture events such as cache misses; hence they are never completely the same. (2) Even though the 4th iteration is from the evaluation of a different statement, there are still similarities between 4th iteration and the other four iterations due to the similar operations performed outside of the selection statements; and hence they are not completely different. Based on these and the given problem definition, there are a few key points that make this a challenging problem:

- The operations in selection statements may have infinite variations since they are used in the application code according to the program needs. Hence, there are infinite patterns that can be observed in a collected side-channel signal, which makes it impossible to learn all beforehand.
- The execution of the same operations is not always the same due to different inputs and possible different architecture events such as interrupts and cache events.
- In most of the cases, it is not possible to obtain a prior information regarding the program code including labeling. Hence, an approach that requires labeled training data would not be applicable in these problems.

All these challenges lead us to propose a new, generalized approach to detect branching statements via dissimilarities between a number of observed signals without requiring prior information such as training data and labeling, which makes the introduced approach more useful and applicable to various scenarios.

In our work, we consider a signal $\boldsymbol{x}$, that can be written as a sequence of T iterations $\boldsymbol{x} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_T]$. Here, each iteration $\boldsymbol{x}_i$ may result in a different number samples N_i such that $\boldsymbol{x}_i = [x_{i,1}, x_{i,2}, \ldots, x_{i,N_i}]$. In this setting, our aim

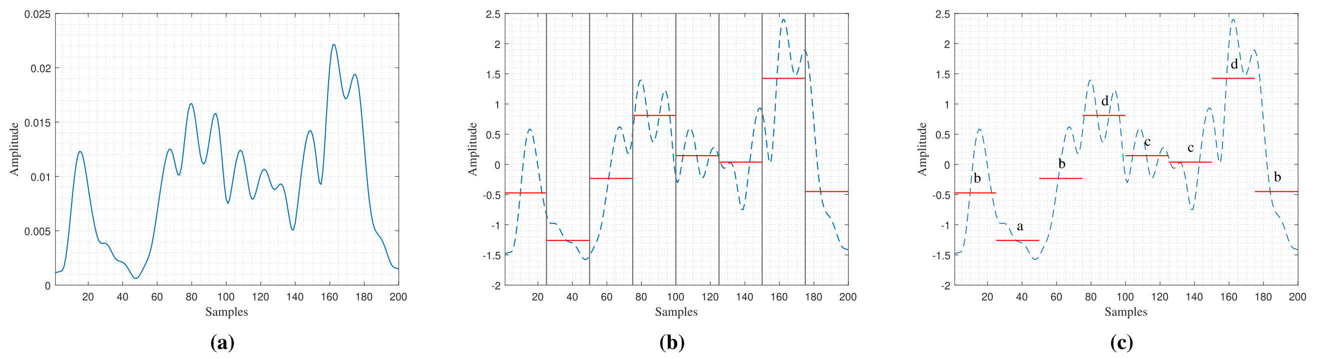


FIGURE 2. Representation steps of the original signal: (a) Single frame of the collected signal, (b) PAA representation of the signal after normalization, (c) SAX representation where the word '*babdccdb*' is constructed from the signal.

is to detect the iterations and samples that are constructed by a different statement and hence showing a different behavior. For this, we propose to use the first iteration of the observed signal as a reference for the remaining. We mainly construct a code-book $\mathcal{M}$, i.e., a set, that contains the patterns existing in the reference signal portion. Then, we compare each newly observed set of samples with these patterns to declare any differentiation. Moreover, we perform these with the signal representations where we map the original long sequences to compact symbolic representations to reduce the computational complexity.

IV. METHOD

In this section, we describe the proposed approach to detect dissimilarities in time series data in various settings. In our framework, we apply a time series representation method to transform and process the observed data instances in a fast and efficient way as in Section IV-A. Then, we use the new representations to compare and detect the dissimilar regions among them as discussed in Section IV-B.

A. TIME SERIES REPRESENTATION

Time series refers to the set of measurements collected from a system or an environment over a time interval. In general, it is collected to model or analyze a system behavior; and hence, it is usually observed as a long sequence with large number of data instances based on the sampling rate or the duration of the investigated process. However, its high-dimensional nature constitutes a challenge for many applications requiring fast processing with low computational complexity [40]. Time series representation is the method of choice in these cases to reduce the data dimension.

For a time series $\boldsymbol{x} \in \mathbb{R}^N$ with N number of data instances, a representation model $R(\cdot)$ constructs a new sequence as

$$R(\boldsymbol{x}) = \bar{\boldsymbol{x}},$$

where $\bar{\boldsymbol{x}} \in \mathbb{R}^n$ with $n \ll N$. Such transformation do not only reduce the computational time but also emphasize the fundamental shape characteristics, which make it particularly useful for pattern analysis applications.

Piecewise aggregate approximation (PAA) is one of the main representation techniques where the observed

TABLE 1. A lookup table for breakpoints with alphabet size α from 3 to 8.

β \ α	3	4	5	6	7	8
β_1	-0.43	-0.67	-0.84	-0.97	-1.07	-1.15
β_2	0.43	0	-0.25	-0.43	-0.57	-0.67
β_3		0.67	0.25	0	-0.18	-0.32
β_4			0.84	0.43	0.18	0
β_5				0.97	0.57	0.32
β_6					1.07	0.67
β_7						1.15

sequence is divided into equal length frames and each frame is represented with the mean of its corresponding data instances [42]. Symbolic aggregate approximation (SAX) is another technique introduced as an extension to the PAA method where the segment mean values are mapped to the alphabet letters [43]. Since it converts the real-valued time series into strings, it also allows one to benefit from text retrieval techniques and text-based algorithms.

Symbolic aggregate approximation method is particularly used in motif discovery and time series classification studies due to its power on capturing signal shape efficiently via discretization. In our study, we take advantage of representation power of this approach to construct a codebook with patterns existing in the observed signal. In our setting, we use it with a sliding window where the representation is applied to every window of the observed signal. For a windowed signal $\boldsymbol{x} = [x_1, x_2, \ldots, x_N]$, PAA representation corresponds to $\bar{\boldsymbol{x}} = [\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n]$ with

$$\bar{x}_j = \frac{n}{N} \sum_{i=\frac{N}{n}(j-1)+1}^{\frac{N}{n}j} x_i.$$

Here, the signal $\boldsymbol{x}$ is first divided into n segments, and each segment is represented with the mean of $\frac{N}{n}$ samples in it to construct $\bar{\boldsymbol{x}}$. Then, the SAX representation converts $\bar{\boldsymbol{x}} \in \mathbb{R}^n$ to the word $\tilde{X} = \tilde{x}_1\tilde{x}_2 \ldots \tilde{x}_n$ based on the set of breakpoints $\mathcal{B} = \{\beta_1, \beta_2, \ldots, \beta_{\alpha-1}\}$ as in Table 1 where α is the alphabet size. The idea of this representation is that the standardization process rescales the amplitude so that the resulting signal have the properties of Gaussian distribution with $N(0, 1)$. Hence, separating the signal in y-axis from the breakpoints in $\mathcal{B}$ divides it into α equiprobable regions. Then, each region

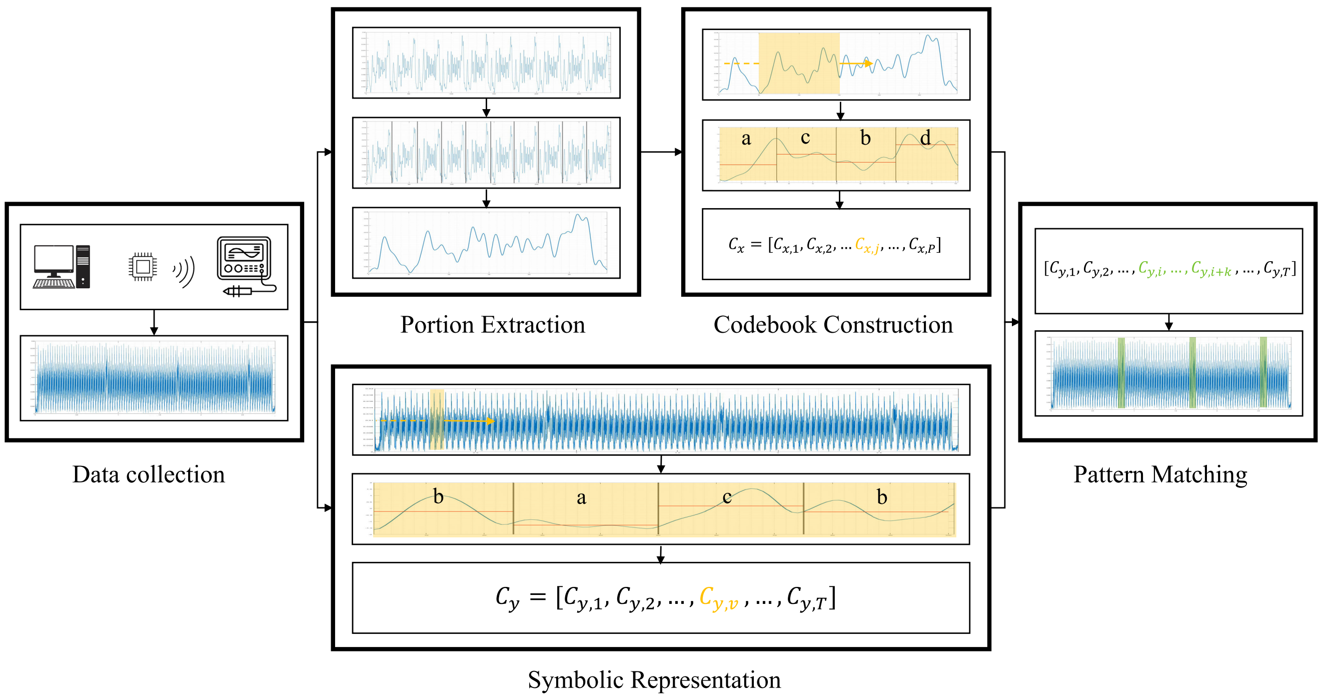

Data collection
Portion Extraction
Codebook Construction
a c b d
$C_x = [C_{x,1}, C_{x,2}, \dots C_{x,j}, \dots, C_{x,P}]$
Symbolic Representation
b a c b
$C_y = [C_{y,1}, C_{y,2}, \dots, C_{y,v}, \dots, C_{y,T}]$
Pattern Matching
$[C_{y,1}, C_{y,2}, \dots, C_{y,i}, \dots, C_{y,i+k}, \dots, C_{y,T}]$

FIGURE 3. Diagram of the proposed method with an application on the EM side-channel signal.

is represented with a symbol that will be assigned to the representation $\bar{\boldsymbol{x}}$ to construct the word $\tilde{X}$.

We provide step-by-step explanation of the proposed method on an example in Fig. 2. In this example, Fig. 2a shows one frame of the original observed signal where the window size is 200 samples. This signal is divided into 8 segments after standardization, and each segment is discretized with their means as shown with red lines in Fig. 2b. In this example, the alphabet size is chosen as $\alpha = 4$. Hence, the symbols $\{a, b, c, d\}$ are assigned to the corresponding regions by starting from the lowest region. As a result, the signal frame is converted into the word '*babdccdb*' in Fig. 2c.

B. EFFICIENT DISSIMILARITY DETECTION

In this section, we present our step-by-step approach to the dissimilarity detection problem.

We observe a time series signal $\boldsymbol{x} = [x_1, x_2, \dots, x_N]$ with N samples that consists of repetitive T patterns. Hence, it can be represented as $\boldsymbol{x} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_T]$ where each iteration $\boldsymbol{x}_i$ is also a sequence of consecutive samples. Since this is a side-channel measurement, even though the i^{th} and j^{th} iterations belong to the same statement, they are not exactly the same. Similarly, for the i^{th} and k^{th} iterations of the same statement, $\boldsymbol{x}_i$ and $\boldsymbol{x}_k$ are not completely different but there are some dissimilar regions between them. In some cases, these iterations may be detected via visual inspection. For the cases where it is not possible to separate the signal iterations, a loop detection method [20] can be applied to extract an iteration. In our setting, we take one iteration (pattern) $\boldsymbol{y} = \boldsymbol{x}_i$ or multiple iterations (a set of l consecutive patterns) $\boldsymbol{y} = [\boldsymbol{x}_i, \boldsymbol{x}_{i+1}, \dots, \boldsymbol{x}_{i+l}]$ to construct a reference. For this, we map $\boldsymbol{y} = [y_1, y_2, \dots, y_M]$ to a sequence of strings $C_{\boldsymbol{y}} = [C_1, C_2, \dots, C_m]$ with the time series representations as described in Section IV-A to construct a codebook containing all observed patterns. We describe this step in detail in Section IV-B1.

1) CODEBOOK CONSTRUCTION

We process the normalized time series $\boldsymbol{y} = [y_1, y_2, \dots, y_M]$ of M samples via sliding window of w samples to capture small patterns efficiently. We refer to the windowed signal at step i as $\boldsymbol{y}_{w_i} = [y_i, y_{i+1}, \dots, y_{i+w-1}]$. We apply the piecewise aggregate approximation and symbolic aggregate approximation representations with the number of segments n and alphabet size α. This mapping $R(\cdot)$ converts the windowed segment $\boldsymbol{y}_{w_i}$ into a string of n symbols such that

$$R(\boldsymbol{y}_{w_i}) = C_i,$$

with

$$C_i = c_i c_{i+1} \dots c_{i+n-1},$$

where c_i is a symbol from the defined alphabet that is assigned based on the breakpoints as in Section IV-A. Such conversion is applied to the complete signal via sliding window and all strings are stored in $C_{\boldsymbol{y}}$ in a sequential manner such that

$$C_{\boldsymbol{y}} = [C_1, C_2, \dots, C_m],$$

where $m = M - w$. Moreover, all observed strings (patterns) and their number of occurrences in $C_{\boldsymbol{y}}$ are stored in a

Algorithm 1 Codebook Construction Algorithm

```
1:  Observe input $\boldsymbol{x} \in \mathbb{R}^T$
2:  Set alphabet size $\alpha$, number of segments $n$, sliding window size $w$
3:  Initialize breakpoints $\mathcal{B} = \{\beta_1, \beta_2, \dots, \beta_{\alpha-1}\}$ in Table 1

4:  Initialize an empty map for codebook $\mathcal{M}_{\boldsymbol{x}}$
5:  for $i = 1, 2, \dots, T - w$ do
6:      Initialize an empty string $C_i$
7:      Define $\boldsymbol{x}^i \leftarrow \boldsymbol{x}[i, i+1, \dots, i-w-1]$
8:      Normalize $\boldsymbol{x}^i$
9:      Separate $\boldsymbol{x}^i$ into $n$ segments $\{\boldsymbol{x}^{i,1}, \boldsymbol{x}^{i,2}, \dots, \boldsymbol{x}^{i,n}\}$
10:     for $j = 1, 2, \dots, n$ do
11:         Calculate mean $\mu_j$ of $\boldsymbol{x}^{i,j}$
12:         Compare $\mu_j$ with the breakpoints $\mathcal{B}$ to find the corresponding symbol $c_j$
13:         Add $c_j$ at the end of $C_i$
14:     end for
15:     Declare string $C_i = c_i c_{i+1} \dots c_{i+n-1}$ of $\boldsymbol{x}^i$ and store in $C_{\boldsymbol{x}}$
16:     if $C_i$ do not exist in $\mathcal{M}_{\boldsymbol{x}}$ then
17:         Initialize $\mathcal{M}_{\boldsymbol{x}}(C_i) = 1$
18:     else
19:         Update $\mathcal{M}_{\boldsymbol{x}}(C_i) = \mathcal{M}_{\boldsymbol{x}}(C_i) + 1$
20:     end if
21: end for
22: Output the sequence $C_{\boldsymbol{x}} = [C_1, C_2, \dots, C_{T-w}]$ and the mapping $\mathcal{M}_{\boldsymbol{x}}$
```

codebook $\mathcal{M}_{\boldsymbol{y}}$. Note that we use the codebook $\mathcal{M}_{\boldsymbol{y}}$ as fixed in our method. However, it can also be updated in an online learning framework to include each newly observed pattern to the sequence to improve the performance.

2) PATTERN MATCHING

Pattern matching refers to comparing two patterns to determine if they are the same or not. Its flexibility can be defined based on user needs as either exact or approximate matching. In this work, we use exact pattern matching approach on strings that are transformed from real valued measurements. We mainly decide that the reference string $C_r = c_{r,1}c_{r,2}\dots c_{r,m}$ and windowed signal representation $C_y = c_{y,1}c_{y,2}\dots c_{y,m}$ match if they are exactly the same $C_r = C_y$ such that

$$\begin{aligned} c_{r,1} &= c_{y,1} \\ c_{r,2} &= c_{y,2} \\ &\dots \\ c_{r,m} &= c_{y,m}. \end{aligned}$$

Otherwise, we flag the location of $\boldsymbol{y} = [x_i, x_{i+1}, \dots, x_{i+l}]$ in the original signal $\boldsymbol{x}$, i.e., $[i, i+1, \dots, i+l]$ as possible dissimilar region.

The algorithm outputs the flagged regions $R = \{\boldsymbol{r}_1, \boldsymbol{r}_2, \dots, \boldsymbol{r}_d\}$ where each region $\boldsymbol{r}$ contains consecutive

Algorithm 2 Dissimilarity Detection Algorithm

```
1:  Observe inputs $\boldsymbol{x}_1 \in \mathbb{R}^{T_1}$ and $\boldsymbol{x}_2 \in \mathbb{R}^{T_2}$
2:  Set alphabet size $\alpha$, number of segments $n$, sliding window size $w$
3:  Set threshold $\tau = p/2$ where $p$ is signal period
4:  Use Algorithm 1 to construct codebook of $\boldsymbol{x}_1$ as $\mathcal{M}_{\boldsymbol{x}_1}$ and the sequence of $\boldsymbol{x}_2$ as $C_{\boldsymbol{x}_2} = [C_1, C_2, \dots, C_{T_2-w}]$
5:  Find the number of occurrences of the strings of $C_{\boldsymbol{x}_2}$ in $M_{\boldsymbol{x}_1}$ and store in $Q = [q_1, q_2, \dots, q_{T_2-w}]$
6:  Flag consecutive samples with all $\{q_t, q_{t+1}, \dots, q_{t+z}\}$ as $\boldsymbol{q}^f$ and store in $Q^f$
7:  Declare flagged region samples $Q^f = \{\boldsymbol{q}_1^f, \boldsymbol{q}_2^f, \dots, \boldsymbol{q}_k^f\}$ where $\boldsymbol{q}_i^f$ and $\boldsymbol{q}_j^f$ are at least $\tau$ samples apart
8:  for $t = 1, 2, \dots, k$ do
9:      Assign initial and end sample indices of $\boldsymbol{q}_t^f$ as $[f_i, f_e]$
10:     Store $s_i = f_i + (n-1) * w/n$ and $s_e = f_e + w/n$ in $S$
11: end for
12: Output $S = \{[s_{1,i}, s_{1,e}], [s_{2,i}, s_{2,e}], \dots, [s_{r,i}, s_{r,e}]\}$ set of dissimilar region indices
```

sample indices flagged as a result of matching step. Then, we decide if a flagged region, i.e., sequence, $\boldsymbol{r}_i$ with length d_i, i.e., the number of consecutive flagged samples, is a dissimilar region via thresholding. The algorithm compares the sequence length d_i with a threshold D and declares $\boldsymbol{r}_i$ as a dissimilar region if $d_i \geq D$. The idea behind this step is that the selection statements perform different consecutive operations, which result in collective variations in the resulting side-channels. Since we apply the symbol representations and pattern matching via sliding window, the flagged samples should be observed consecutively in a similar manner.

The illustration of the end-to-end process is given in Fig. 3. The collected signal is first processed to extract a portion such as one or multiple iterations to represent a reference pattern. This can be applied either via visual inspection or period detection algorithm [20] in the portion extraction step. Then, the extracted portion is processed with a sliding window where each frame is transformed into its symbolic representation. As a result, the codebook that contains observed patterns is constructed to be used as a reference. In the meantime, the whole signal is processed via sliding window in a similar way to construct a sequence of patterns in their symbolic form. In the pattern matching step, the symbolic representation of the signal is compared with the constructed codebook, and the consecutive newly observed patterns are declared as dissimilar regions.

We provide the codebook construction algorithm that we use to transform the observed signals into their symbolic representations in Algorithm 1. The algorithm processes the observed input $\boldsymbol{x}$ by sliding window. At each time, the input frame is normalized and separated into n segments. Each segment is transformed into a symbol based on comparison of its mean with the breakpoints $\mathcal{B}$ as in Table 1. After transforming each segment, a string that represents the input frame is constructed. At the end of this process, the input signal is represented with a sequence of strings and all observed patterns along with their number of occurrences are stored.

We provide the dissimilarity detection algorithm that outputs the dissimilar regions based on pattern matching in Algorithm 2. The algorithm uses the codebook $\mathcal{M}_{\boldsymbol{x}_1}$ of the reference input $\boldsymbol{x}_1$ and the symbolic representation sequence $C_{\boldsymbol{x}_2}$ of the main input $\boldsymbol{x}_2$ that are processed and generated separately via Algorithm 1. The patterns are compared and consecutive z samples of $\boldsymbol{x}_2$ that do not exist in the codebook $\mathcal{M}_{\boldsymbol{x}_1}$ are flagged where $z \geq \tau$ and τ is pre-defined threshold. Then, the corresponding sample locations in the original signal are stored in S as beginning and end indices of the dissimilar regions $[s_i, s_e]$. Finally, the algorithm outputs all detected dissimilar regions along with their sample indices.

V. EXPERIMENTS

We perform experiments on practical data to show the performance of the proposed method. We first describe the experimental setup that is used to collect the measurements. Then, we provide the details of each experiment along with the performance results.

A. EXPERIMENTAL SETUP

We use two devices BeagleBone and A13-OLinuXino to collect electromagnetic (EM) side-channel signals for our experiments as in Fig. 4. BeagleBone Black single-board computer has ARM Cortex-A8 processor whose operating frequency is 1 GHz. Similarly, A13-OLinuXino is a single-board embedded Linux computer that has ARM Cortex-A8 processor with 1 GHz operating clock frequency.

In the experiments, we run various application implementations on the boards where we particularly use *bitcount* algorithm of MiBench [46] benchmark suite as a real-world application. We provide the details of the implementations run on the devices in Section V-B and V-C along with their experimental results.

We collect the EM signals emanating from the devices with near-field magnetic probes located around the device processors as in Fig. 4. For BeagleBone and A13-OLinuXino boards, we use EMC H5 and Aaronia H2 probes, respectively. For recording of the collected signals, we use Keysight UXA signal analyzer with 1.28 GHz sampling rate.

In the pre-processing step, we filter the signals collected from BeagleBone for interference removal. However, the signals of A13-OLinuXino device are directly used without any pre-processing since it does not introduce any significant interference.

B. SYNTHETIC CODE EXPERIMENTS

To illustrate the performance and applicability of the discussed approach, we first construct a program code that

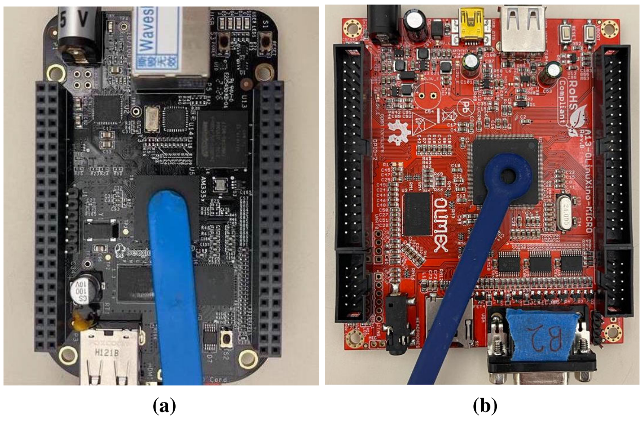


FIGURE 4. Measurement setup used to collect emanating EM signals for (a) BeagleBone Black, (b) A13-OLinuXino devices.

can be observed in various applications. For this, we use the following code structure to run on the device:

```
for (int i = 0; i < N; ++i) {
  int flag = flagArray(i);
  if (flag == 1)
    n = statement_set_1 (in1, ...,
        inK);
  else
    n = statement_set_2 (in1, ...,
        inK);
}
```

where `statement_set_1` and `statement_set_2` refer to the different sets of arithmetic operations that are performed consecutively by the processor. For each iteration `i`, the algorithm evaluates the defined selection statement and chooses one of the paths based on the resulting value of the `flag` variable. In this example, the algorithm iterates 30 times and performs operations in `statement_set_1` if `i`$\in$ {10, 12, 16, 18}. Otherwise, it follows operations in `statement_set_2` to assign the result to `n`. We run this algorithm on BeagleBone device and collect the EM side-channel measurements as shown in Fig. 4a.

We obtain the signal given in Fig. 5c with 30 iterations. First two iterations shown in Fig. 5a are fed to the algorithm to construct a reference codebook. Then, complete signal is processed and compared with the constructed reference patterns. As a result, the number of pattern occurrences is obtained as in Fig. 5b. Here, the block of patterns that do not exist in the reference codebook are shown as green regions. The algorithm successfully detects the corresponding dissimilar regions highlighted with green in Fig. 5c. These regions exactly correspond to the 10, 12, 16 and 18th iterations where the algorithm performs operations in `statement_set_1` which results in different patterns compared to the remaining signal.

Note that the signal blocks of two iterations shown in Fig. 5a are the results of the same set of operations, and they are not exactly the same. This is an expected result due to the nature of the side-channels and the investigated problem,

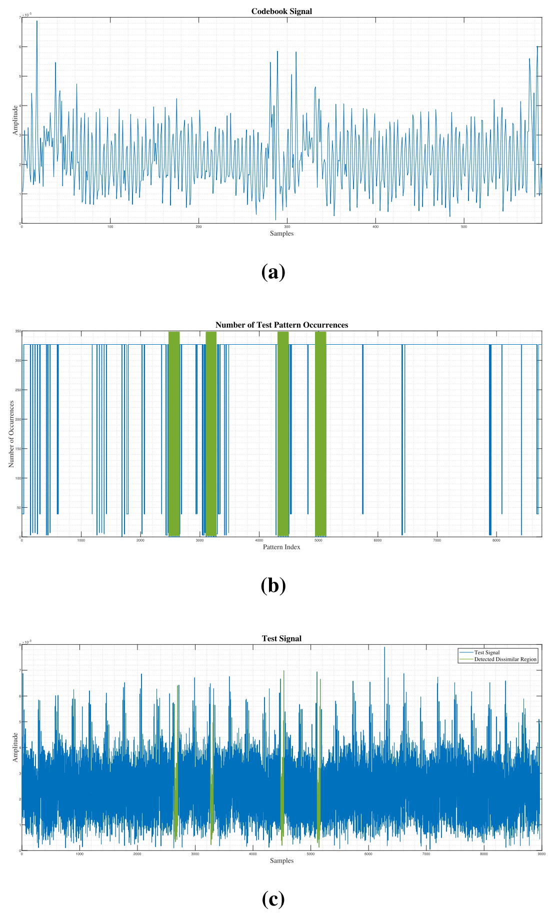


FIGURE 5. Side-channel measurement with the BeagleBone device: (a) Two iterations of the collected signal that is used to construct codebook, (b) Number of occurrences of the signal patterns based on the pattern codebook where the highlighted regions are the detected dissimilarities, (c) 30 iterations of the second measurement where the green regions refer to the dissimilar regions correctly detected by the algorithm.

which makes it more challenging as discussed in Section III. However, even with this challenging signal, the constructed reference patterns successfully represent the signal behaviors and the proposed method efficiently finds the dissimilar regions caused by the selection statements.

C. REAL APPLICATION CODE EXPERIMENTS

In this section, we perform experiments to show the performance of the proposed method on real-world applications. For this, we use MiBench [46] benchmark suite and particularly its *bitcount* algorithm. MiBench consists of benchmarks of various applications that offer different program characteristics. Among these, the category of Automotive and Industrial Control represents the use of embedded processors in embedded control systems. Its applications include engine performance monitoring, air bag controllers and sensor systems, which require bit manipulation, basic math abilities and data organization. One of the important algorithms in this category is *bitcount*, which tests the bit manipulation abilities of a processor by counting the number of bits in given integers. It contains the implementations of optimized 1-bit per loop counter, recursive bit count by nibbles, non-recursive bit count by nibbles with a look-up table, non-recursive bit count by bytes and shift and count bits [46]. The main loop of *bitcount* algorithm benchmark is given below

```
for (i = 0; i < FUNCS; i++) {
  ...
  for (j = n = 0, seed = rand(); j <
      iterations; j++, seed += 13)
    n += pBitCntFunc[i](seed);
  ...
}
```

where `pBitCntFunc` calls the *bitcount* methods in the following order

```
static int (* CDECL
     pBitCntFunc[FUNCS])(long) = {
  bit_count,
  bitcount,
  ntbl_bitcnt,
  ntbl_bitcount,
  BW_btbl_bitcount,
  AR_btbl_bitcount,
  bit_shifter
};
```

In our setting, we use the *bitcount* algorithm due to its wide application areas. We construct two experiment setups with *bitcount* implementations to present performance results in different applications separately.

1) APPLICATION SETUP 1

The method of "recursive bit count by nibbles", i.e., `ntbl_bitcnt` function, in *bitcount* contains a selection statement to control the recursive calls. The main structure of this method is as follows

```
int CDECL ntbl_bitcnt(long x)
{
  int cnt = bits[(int)(x & 0x0000000FL)];
  if (0L != (x >>= 4))
    cnt += ntbl_bitcnt(x);
  return cnt;
}
```

where the bit count is performed in a recursive manner through the function calls inside the `if()` selection statement. In our experiment, we aim to detect the dissimilarities due to this selection statement and their locations on the measurement data via our proposed approach.

We run the *bitcount* algorithm on A13-OlinuXino device and collect the EM side-channels emanating during program running with the setup in Fig. 4b. Note that the number of recursions in the output signal varies based on the given input `seed` and its increments. In our setting, we increment `seed` by `1M` to observe the variations in the output signal in a limited duration. As a result, we observe 10 out of 30 iterations of the collected signal as in Fig. 6c. Note that all

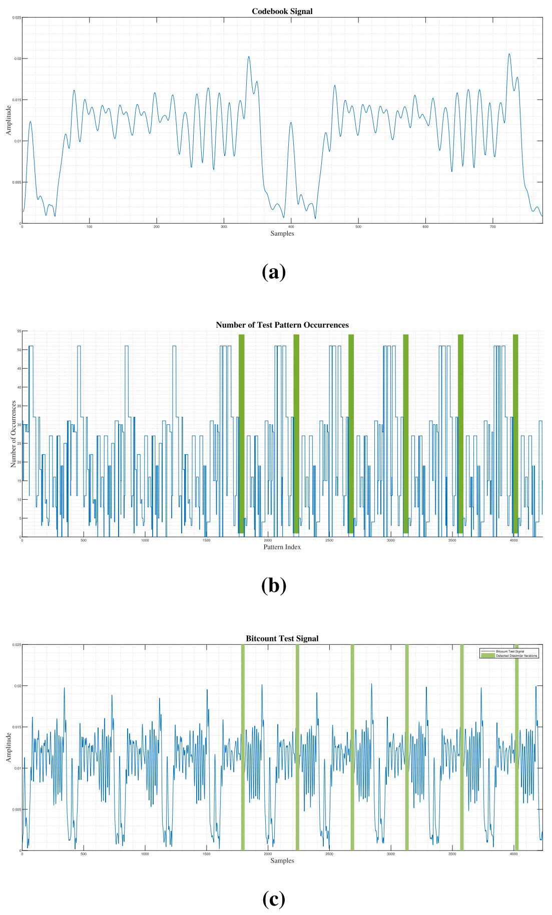


FIGURE 6. Side-channel measurement of the *bitcount* code: (a) First measurement with two iterations used to construct codebook, (b) Number of occurrences of the patterns of the second measurement signal in the pattern codebook where the highlighted regions are the detected dissimilarities, (c) 10 iterations of the second measurement where green lines refer to the dissimilar iterations correctly detected by the algorithm.

iterations (even the ones with the same number of iterations) are different from each other due to the processor of the device and the other effects such as noise as discussed in previous sections. However, the main difference is caused by the selection statement where the first 4 loop iterations contain 6 recursive calls while the remaining ones contain 7 recursive calls of the `ntbl_bitcount` function.

To use our proposed method, we collect two versions of the signal, and we use only two iterations of one of them as in Fig. 6a to construct the codebook. We apply our method on the other signal and compare its patterns with the patterns in the codebook. We obtain the number of pattern occurrences as in Fig. 6b for the first 10 iterations. As seen from this figure, even though there are similar patterns between the main signal and the codebook, the dissimilar regions are detected as consecutive unmatched regions (highlighted in green). Similarly, we present the results belonging to 10 iterations in Fig. 6c where the algorithm successfully detects that dissimilarity that exists in all iterations after the first 4 iterations in the beginning.

2) APPLICATION SETUP 2

As described in Section V-C, *bitcount* algorithm contains seven methods with various implementations. In this set of experiments, we combine two of these methods via selection statement in a loop structure, which represent the implementation of various real applications. For this, we modify the main loop of *bitcount* as

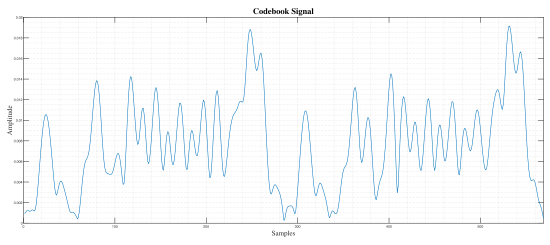
```
for (i = 0; i < FUNCS; i++) {
  ...
  for (j = n = 0, seed = rand(); j <
      iterations; j++, seed += 13){
    if (j % K == 0)
      n += ntbl_bitcnt[i](seed);
    else
      n += ntbl_bitcount[i](seed);
    }
  ...
}
```

such that it calls either one of the implementations of bit count by nibbles, i.e., `ntbl_bitcnt` or `ntbl_bitcount` based on the evaluation of `if()` statement. The only difference between these functions is that `ntbl_bitcnt` is recursive while `ntbl_bitcount` is a non-recursive implementation with a look-up table.

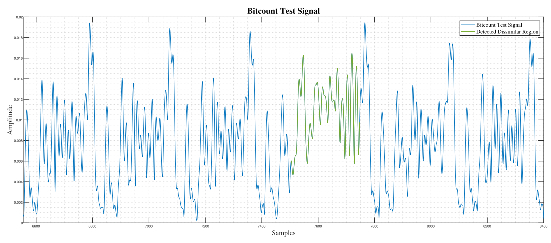
To collect the side-channel measurements, we run the code on A13-OlinuXino device as in Fig. 4b. We fix $K = 30$ in the application implementation and the number of iterations as 100 for illustration purposes where only 3% of the loop iterations call a different function and perform different operations. Such behavior can be seen in various real scenarios such as fault check/input check points which are placed inside the code where the program operates differently if the pre-defined condition in a selection statement is not met. In real applications where it is not possible to control the algorithm, such faulty behavior can crash the running program and one cannot foresee and learn patterns beforehand. Therefore, in such cases, it is crucial to understand the code behavior and detect the fault in the form of dissimilarity by processing the measurements without interrupting the process.

We use the two iterations of the collected signal in Fig. 7a with only 570 samples to construct the codebook. Then, we apply the proposed algorithm on the complete signal with approximately 2.77×10^4 samples to compare their patterns. For illustration, we provide only a portion of the processed signal in Fig. 7b. In this figure, the detected consecutive dissimilar samples are also highlighted with red. Note that even the dissimilar iterations share the same patterns in the beginning and end due to other operations performed before and after the functions called in selection statements. The proposed algorithm correctly identifies the previously seen patterns and detects the dissimilar sample collections as shown in Fig. 7b.

In this setup, the resulting signal contains 682 samples belonging to the three separate dissimilar regions (due to operations in `if()` statement) and 27007 samples of the



(a)



(b)

FIGURE 7. Side-channel measurement of the *bitcount* code with application setup 2: (a) Measurement with two iterations used to construct codebook, (b) 6 iterations of the measurement where green samples correspond to the dissimilar collective samples detected by the algorithm.

similar regions (due to operations of `else`), i.e., similar patterns with the codebook. The codebook is constructed with the initial two iterations (with 570 samples), and the algorithm is run with alphabet size $\alpha = 3$, the number of segments $n = 5$ and window size $w = 300$. The detection of dissimilar patterns is decided based on thresholding the number of consecutive dissimilarities as discussed in Section IV. To show the performance of the method, we plot the Receiver Operating Characteristic (ROC) curve by changing this threshold value and obtain Area Under Curve (AUC) as 0.994 with F_1 score = 0.980.

Note that, we mainly propose an approach that can be used to detect dissimilar regions among multiple observations or single time series with repetitive patterns. In this framework, the information on the observed patterns and their number of occurrences that we obtained as a result of our approach as in previous examples provides a useful and comprehensive insight regarding the possible dissimilarity locations. In addition to this, our proposed algorithm can also be used to declare the flagged regions as sample indices. For a comprehensive analysis, in this section, we also change the dissimilarity percentage to approximately 20% of loop iterations. Since our method do not have any assumption on the distribution such as rarety, it achieves the similar performance results with the same parameters compared to 3% dissimilarity. In this case, we run the algorithm for additional various alphabet size α, number of segments n and window length w combinations and provide the AUC results obtained by changing the threshold in Table 2. As seen from the results, the algorithm achieves high AUC values in all cases which shows its high detection performance.

TABLE 2. The AUC results of the algorithm with various parameter combinations on signals collected with application setup 2.

Alphabet Size (α)	# of Segments (n)	Window Size (w)	AUC
3	5	300	0.994
5	5	300	0.987
8	5	300	0.982
3	3	300	0.984
3	10	300	0.998
3	5	200	0.849
3	5	400	0.982

In addition, we observe that increasing α while keeping the other parameters same slightly decreases the performance since it increases the number of horizontal segmentation, which decreased the tolerance to the environmental effects on the measurement such as noise. Moreover, increasing the number of segments n results in more detailed modelling and better performance. Finally, we observe the effect of window size w on the performance. In this example, the signal contains repeating iterations where each of iteration has approximately 300 samples. Therefore, using a window size w similar to or greater than this value provides a better performance since the dominant pattern can be modelled better and included in the codebook.

VI. CONCLUSION

In this paper, we have proposed a dissimilarity detection algorithm that can find different regions between similar signals. We particularly addressed the detection problem of branching statements in program control flows via side-channels where we showed that such statements cause dissimilarities in emanating side-channels and we propose a generalized approach that can be efficiently applied to various real-world applications. In the introduced method, symbolic representations of the signals are used for efficient processing where each signal frame is transformed into a string. The codebook of observed patterns is constructed with the reference signal. Then, the sequence of strings that is obtained from the main signal is compared with the codebook to find the newly observed patterns. Finally, the presented method outputs the samples of dissimilar regions in the signal compared to the reference. In the experiments, we provided a comprehensive analysis on various cases of real applications with EM side-channel signals collected from different devices to show the applicability and efficiency of the proposed method and the results show that dissimilarities can be detected with $> 98\%$ accuracy.

REFERENCES

[1] S. Li, L. Xu, and S. Zhao, "The Internet of Things: A survey," *Inf. Syst. Frontiers*, vol. 17, no. 2, pp. 243–259, 2015.

[2] F. Ullah, H. Naeem, S. Jabbar, S. Khalid, M. A. Latif, F. Al-Turjman, and L. Mostarda, "Cyber security threats detection in Internet of Things using deep learning approach," *IEEE Access*, vol. 7, pp. 124379–124389, 2019.

[3] R. Callan, F. Behrang, A. Zajic, M. Prvulovic, and A. Orso, "Zero-overhead profiling via EM emanations," in *Proc. 25th Int. Symp. Softw. Test. Anal.* New York, NY, USA: Association for Computing Machinery, Jul. 2016, pp. 401–412.

[4] H. Kim, M. A. Suleman, O. Mutlu, and Y. N. Patt, "2D-profiling: Detecting input-dependent branches with a single input data set," in *Proc. Int. Symp. Code Gener. Optim. (CGO)*, Mar. 2006, pp. 1–11.

[5] B. B. Yilmaz, E. M. Ugurlu, F. Werner, M. Prvulovic, and A. Zajic, "Program profiling based on Markov models and EM emanations," *Proc. SPIE*, vol. 11417, Apr. 2020, Art. no. 114170D.

[6] H. A. Khan, M. Alam, A. Zajic, and M. Prvulovic, "Detailed tracking of program control flow using analog side-channel signals: A promise for IoT malware detection and a threat for many cryptographic implementations," *Proc. SPIE*, vol. 10630, Mar. 2018, Art. no. 1063005.

[7] Y. Liu, L. Wei, Z. Zhou, K. Zhang, W. Xu, and Q. Xu, "On code execution tracking via power side-channel," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*. New York, NY, USA: Association for Computing Machinery, Oct. 2016, pp. 1019–1031.

[8] H. A. Khan, N. Sehatbakhsh, L. N. Nguyen, M. Prvulovic, and A. Zajić, "Malware detection in embedded systems using neural network model for electromagnetic side-channel signals," *J. Hardw. Syst. Secur.*, vol. 3, no. 4, pp. 305–318, Dec. 2019.

[9] Y. Pan, X. Ge, C. Fang, and Y. Fan, "A systematic literature review of Android malware detection using static analysis," *IEEE Access*, vol. 8, pp. 116363–116379, 2020.

[10] S. S. Clark, B. Ransford, A. Rahmati, S. Guineau, J. Sorber, W. Xu, and K. Fu, "WattsUpDoc: Power side channels to nonintrusively discover untargeted malware on embedded medical devices," in *Proc. USENIX Workshop Health Inf. Technol. (HealthTech)*. Washington, DC, USA: USENIX Association, Aug. 2013, pp. 1–11.

[11] K. Ramezanpour, P. Ampadu, and W. Diehl, "SCAUL: Power side-channel analysis with unsupervised learning," *IEEE Trans. Comput.*, vol. 69, no. 11, pp. 1626–1638, Nov. 2020.

[12] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi, "The EM side—Channel(s)," in *Cryptographic Hardware and Embedded Systems—CHES 2002*, B. S. Kaliski, Ç. K. Koç, and C. Paar, Eds. Berlin, Germany: Springer, 2003, pp. 29–45.

[13] D. Genkin, A. Shamir, and E. Tromer, "Acoustic cryptanalysis," *J. Cryptol.*, vol. 30, no. 2, pp. 392–443, Apr. 2017.

[14] C. Lavaud, R. Gerzaguet, M. Gautier, O. Berder, E. Nogues, and S. Molton, "Whispering devices: A survey on how side-channels lead to compromised information," *J. Hardw. Syst. Secur.*, vol. 5, no. 2, pp. 143–168, Jun. 2021.

[15] D. Spatz, D. Smarra, and I. Ternovskiy, "A review of anomaly detection techniques leveraging side-channel emissions," *Proc. SPIE*, vol. 11011, Mar. 2019, Art. no. 110110E.

[16] F. Ding, H. Li, F. Luo, H. Hu, L. Cheng, H. Xiao, and R. Ge, "DeepPower: Non-intrusive and deep learning-based detection of IoT malware using power side channels," in *Proc. 15th ACM Asia Conf. Comput. Commun. Secur. (ASIA CCS)*. New York, NY, USA: Association for Computing Machinery, Oct. 2020, pp. 33–46.

[17] H. A. Khan, N. Sehatbakhsh, L. N. Nguyen, R. L. Callan, A. Yeredor, M. Prvulovic, and A. Zajic, "IDEA: Intrusion detection through electromagnetic-signal analysis for critical embedded and cyber-physical systems," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 3, pp. 1150–1163, May 2021.

[18] R. Rutledge, S. Park, H. Khan, A. Orso, M. Prvulovic, and A. Zajic, "Zero-overhead path prediction with progressive symbolic execution," in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng. (ICSE)*, May 2019, pp. 234–245.

[19] A. Zajic and M. Prvulovic, "Experimental demonstration of electromagnetic information leakage from modern processor-memory systems," *IEEE Trans. Electromagn. Compat.*, vol. 56, no. 4, pp. 885–893, Aug. 2014.

[20] M. Kerpicci, M. Prvulovic, and A. Zajic, "A hierarchical approach for multiple periodicity detection in software code analysis," *IEEE Access*, vol. 10, pp. 106936–106945, 2022.

[21] B. B. Yilmaz, M. Prvulovic, and A. Zajic, "Electromagnetic side channel information leakage created by execution of series of instructions in a computer processor," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 776–789, 2020.

[22] T. Ball and J. R. Larus, "Optimally profiling and tracing programs," *ACM Trans. Program. Lang. Syst.*, vol. 16, no. 4, pp. 1319–1360, Jul. 1994.

[23] N. Kumar, B. R. Childers, and M. L. Soffa, "Low overhead program monitoring and profiling," in *Proc. 6th ACM SIGPLAN-SIGSOFT Workshop Program Anal. Softw. Tools Eng. (PASTE)*. New York, NY, USA: Association for Computing Machinery, Sep. 2005, pp. 28–34.

[24] A. Sayakkara, N.-A. Le-Khac, and M. Scanlon, "Leveraging electromagnetic side-channel analysis for the investigation of IoT devices," *Digit. Invest.*, vol. 29, pp. S94–S103, Jul. 2019.

[25] J. Park, F. Rahman, A. Vassilev, D. Forte, and M. Tehranipoor, "Leveraging side-channel information for disassembly and security," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 16, no. 1, pp. 1–21, Dec. 2019.

[26] A. P. Sayakkara and N.-A. Le-Khac, "Electromagnetic side-channel analysis for IoT forensics: Challenges, framework, and datasets," *IEEE Access*, vol. 9, pp. 113585–113598, 2021.

[27] M. Dey, B. B. Yilmaz, M. Prvulovic, and A. Zajic, "PRIMER: Profiling interrupts using electromagnetic side-channel for embedded devices," *IEEE Trans. Comput.*, vol. 71, no. 8, pp. 1824–1838, Aug. 2022.

[28] A. P. Sayakkara and N.-A. Le-Khac, "Forensic insights from smartphones through electromagnetic side-channel analysis," *IEEE Access*, vol. 9, pp. 13237–13247, 2021.

[29] Y. Han, S. Etigowni, H. Liu, S. Zonouz, and A. Petropulu, "Watch me, but don't touch me! Contactless control flow monitoring via electromagnetic emanations," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*. New York, NY, USA: Association for Computing Machinery, Oct. 2017, pp. 1095–1108.

[30] A. Nazari, N. Sehatbakhsh, M. Alam, A. Zajic, and M. Prvulovic, "EDDIE: EM-based detection of deviations in program execution," *ACM SIGARCH Comput. Archit. News*, vol. 45, no. 2, pp. 333–346, Jun. 2017.

[31] X. Wang, Q. Zhou, J. Harer, G. Brown, S. Qiu, Z. Dou, J. Wang, A. Hinton, C. A. Gonzalez, and P. Chin, "Deep learning-based classification and anomaly detection of side-channel signals," *Proc. SPIE*, vol. 10630, May 2018, Art. no. 1063006.

[32] J.-J. Quisquater and D. Samyde, "ElectroMagnetic Analysis (EMA): Measures and counter-measures for smart cards," in *Smart Card Programming and Security*, I. Attali and T. Jensen, Eds. Berlin, Germany: Springer, 2001, pp. 200–210.

[33] A. A. Goshtasby, "Similarity and dissimilarity measures," in *Image Registration*. London, U.K.: Springer, 2012, pp. 7–66.

[34] H. Jegou, C. Schmid, H. Harzallah, and J. Verbeek, "Accurate image search using the contextual dissimilarity measure," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 1, pp. 2–11, Jan. 2010.

[35] L. Nanni, A. Rigo, A. Lumini, and S. Brahnam, "Spectrogram classification using dissimilarity space," *Appl. Sci.*, vol. 10, no. 12, p. 4176, Jun. 2020.

[36] K. Umapathy, S. Krishnan, and R. K. Rao, "Audio signal feature extraction and classification using local discriminant bases," *IEEE Trans. Audio, Speech Language Process.*, vol. 15, no. 4, pp. 1236–1246, May 2007.

[37] D. Li, Y. Zhao, and Y. Li, "Time-series representation and clustering approaches for sharing bike usage mining," *IEEE Access*, vol. 7, pp. 177856–177863, 2019.

[38] J. Lin and Y. Li, "Finding structural similarity in time series data using bag-of-patterns representation," in *Scientific and Statistical Database Management*, M. Winslett, Ed. Berlin, Germany: Springer, 2009, pp. 461–477.

[39] M. G. Baydogan, G. Runger, and E. Tuv, "A bag-of-features framework to classify time series," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 11, pp. 2796–2802, Nov. 2013.

[40] S. J. Wilson, "Data representation for time series data mining: Time domain approaches," *WIREs Comput. Statist.*, vol. 9, no. 1, p. e1392, Jan./Feb. 2017.

[41] X. Wang, A. Mueen, H. Ding, G. Trajcevski, P. Scheuermann, and E. Keogh, "Experimental comparison of representation methods and distance measures for time series data," *Data Mining Knowl. Discovery*, vol. 26, no. 2, pp. 275–309, Mar. 2013.

[42] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra, "Dimensionality reduction for fast similarity search in large time series databases," *Knowl. Inf. Syst.*, vol. 3, no. 3, pp. 263–286, 2001.

[43] J. Lin, E. Keogh, L. Wei, and S. Lonardi, "Experiencing SAX: A novel symbolic representation of time series," *Data Mining Knowl. Discovery*, vol. 15, no. 2, pp. 107–144, Aug. 2007.

[44] Y. Sun, J. Li, J. Liu, B. Sun, and C. Chow, "An improvement of symbolic aggregate approximation distance measure for time series," *Neurocomputing*, vol. 138, pp. 189–198, Aug. 2014.

[45] D. E. Knuth, J. H. Morris Jr., and V. R. Pratt, "Fast pattern matching in strings," *SIAM J. Comput.*, vol. 6, no. 2, pp. 323–350, Jun. 1977.

[46] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *Proc. 4th Annu. IEEE Int. Workshop Workload Characterization (WWC)*, Dec. 2001, pp. 3–14.

MINE KERPICCI (Graduate Student Member, IEEE) received the B.S. degree from Middle East Technical University, in 2017, and the M.S. degree in electrical and electronics engineering from Bilkent University, Ankara, Turkey, in 2019. She is currently pursuing the Ph.D. degree with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, USA. Her research interests include signal processing, machine learning, and optimization.

MILOS PRVULOVIC (Senior Member, IEEE) received the B.Sc. degree in electrical engineering from the University of Belgrade, in 1998, and the M.Sc. and Ph.D. degrees in computer science from the University of Illinois at Urbana–Champaign, in 2001 and 2003, respectively. He joined the Georgia Institute of Technology, in 2003, where he is currently a Professor with the School of Computer Science. His research interests include computer architecture, especially hardware support for software monitoring, debugging, and security. He was a Senior Member of the ACM and the IEEE Computer Society. He was a recipient of the NSF CAREER Award.

ALENKA ZAJIĆ (Senior Member, IEEE) received the B.Sc. and M.Sc. degrees form the School of Electrical Engineering, University of Belgrade, in 2001 and 2003, respectively, and the Ph.D. degree in electrical and computer engineering from the Georgia Institute of Technology, in 2008. She was a Visiting Faculty Member with the School of Computer Science, Georgia Institute of Technology, a Postdoctoral Fellow with the U.S. Naval Research Laboratory, and a Design Engineer with Skyworks Solutions Inc. She is currently a Ken Byers Professor with the School of Electrical and Computer Engineering, Georgia Institute of Technology. Her research interests include electromagnetic, wireless communications, signal processing, and computer engineering. She was a recipient of the 2017 NSF CAREER Award, the 2012 Neal Shepherd Memorial Best Propagation Paper Award, the Best Student Paper Award from the 2014 IEEE International Conference on Communications and Electronics, the Best Paper Award from the 2008 International Conference on Telecommunications, the Best Student Paper Award from the 2007 Wireless Communications and Networking Conference, and the Dan Noble Fellowship, in 2004, which was awarded by Motorola Inc., and the IEEE Vehicular Technology Society for quality impact in the area of vehicular technology. She is also an Editor of IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS.