Received March 26, 2020, accepted April 10, 2020, date of publication April 15, 2020, date of current version April 30, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2988160

KeySplitWatermark: Zero Watermarking Algorithm for Software Protection Against Cyber-Attacks

CELESTINE IWENDI[1], (Senior Member, IEEE), ZUNERA JALIL[2], (Member, IEEE), ABDUL REHMAN JAVED[3], THIPPA REDDY G.[4], RAJESH KALURI[4], GAUTAM SRIVASTAVA[5,6], (Senior Member, IEEE), AND OHYUN JO[7], (Member, IEEE)

[1]Department of Electronics BCC, Central South University of Forestry and Technology, Changsha 410004, China
[2]Department of Cyber Security, Air University, Islamabad 44000, Pakistan
[3]National Center for Cyber Security, Air University, Islamabad 44000, Pakistan
[4]School of Information Technology and Engineering, VIT, Vellore 632014, India
[5]Department of Mathematics and Computer Science, Brandon University, Brandon, MB R7A 6A9, Canada
[6]Research Center for Interneural Computing, China Medical University, Taichung 40402, Taiwan
[7]Department of Computer Science, College of Electrical and Computer Engineering, Chungbuk National University, Cheongju 28644, South Korea

Corresponding authors: Gautam Srivastava (srivastavag@brandonu.ca) and Ohyun Jo (ohyunjo@chungbuk.ac.kr)

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) under Grant NRF-2018R1C1B5045013.

ABSTRACT Cyber-attacks are evolving at a disturbing rate. Data breaches, ransomware attacks, crypto-jacking, malware and phishing attacks are now rampant. In this era of cyber warfare, the software industry is also growing with an increasing number of software being used in all domains of life. This evolution has added to the problems of software vendors and users where they have to prevent a wide range of attacks. Existing watermark detection solutions have a low detection rate in the software. In order to address this issue, this paper proposes a novel blind Zero code based Watermark detection approach named *KeySplitWatermark*, for the protection of software against cyber-attacks. The algorithm adds watermark logically into the code utilizing the inherent properties of code and gives a robust solution. The embedding algorithm uses keywords to make segments of the code to produce a key-dependent on the watermark. The extraction algorithms use this key to remove watermark and detect tampering. When tampering increases to a certain user-defined threshold, the original software code is restored making it resilient against attacks. *KeySplitWatermark* is evaluated on tampering attacks on three unique samples with two distinct watermarks. The outcomes show that the proposed approach reports promising results against cyber-attacks that are powerful and viable. We compared the performance of our proposal with state-of-the-art works using two different software codes. Our results depict that *KeySplitWatermark* correctly detects watermarks, resulting in up to 15.95 and 17.43 percent reduction in execution time on given code samples with no increase in program size and independent of watermark size.

INDEX TERMS Cyber-attacks, watermarking, software, algorithm, blind detection, attack models, security.

I. INTRODUCTION

In the 21st century, with enormous computational power, high-speed internet, Internet of Things and Blockchain technology, business can be done using Bitcoins. It shows that digital contents are widespread on a wide range of connected devices. Individuals of the current electronic era are sharing information in real-time but at the same time face the problems of evolving cyber-attacks on their data, software, systems, devices, and services. Digital forgeries, cyber frauds, malware, smart bot (DDoS) attacks, software and data breaches are quite common [1].

Digital objects such as software, databases, images, audio, videos, and webpages get created and widely distributed over the internet in no time. Hackers try to break the security layer of the system by exploiting the known software vulnerabilities and then attack using viruses, malware, Trojan horses, logic bombs, backdoors, etc. Software codes are modified by

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

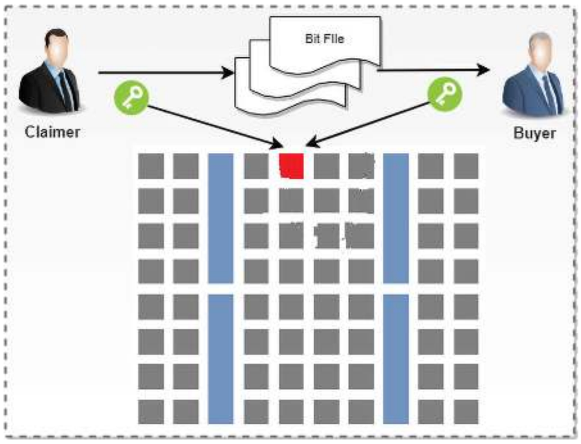

FIGURE 1. Graphical representation of conventional watermarking.

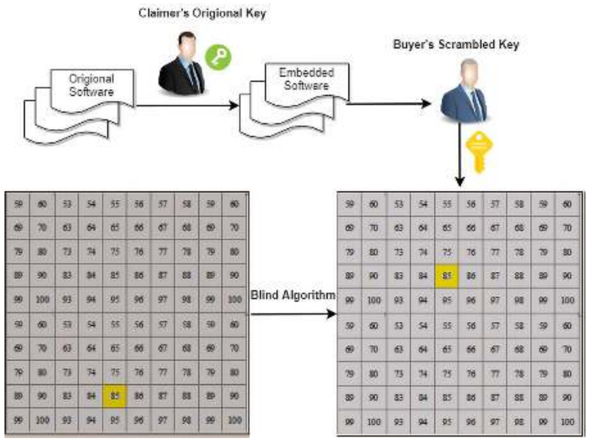

FIGURE 2. Process of zero blind watermarking.

malicious attackers to create malware and cause harm to the software systems. According to Symantec's Internet Security Threat Report, 2018 [2], 99.9% of applications on third-party App stores are malware. Software protection and increasing its resilience is crucial for prevention against cyber-attacks.

Software is an important digital component as it is the foundation of a computer, internet, and all communication infrastructures. Software applications are being developed with a rapid rate for communication, education, commerce, health care, cloud, and many other domains. At the same time software can be attacked and used for totally different or malicious purposes in no time. Common Vulnerabilities and Exposures database [3], [4] shows that more and more new vulnerabilities are getting disclosed with each passing day and today's software is more vulnerable to cyber-attacks. Exploiting vulnerabilities in software code can make it serve a totally different purpose and can be exploited by cybercriminals to perform an attack.

The existing traditional watermarking extraction needs to provide real watermark locations. It poses a great threat to the security of watermarks because it cannot ensure whether the watermark locations will be leaked as shown in Figure 1.

Figure 2 explains the process of watermarking the original source code using a blind Code based Zero Watermark algorithm. The claimer firstly scrambles the source code and the watermark positions are sent to the verifier, who will perform the watermark detection. If the scrambling parameters are not public, the verifier cannot derive the original watermark positions.

The zero watermarking embedding algorithm utilizes [5] the software code's structure and watermark to construct a key. This key must be registered with Certification Authority (CA) along with the original watermark. In case an attack is suspected as per user settings, the extraction algorithm extracts the key from tampered software source code and can identify tampering by matching this key with the CA. In the case of tampering, if the payload is greater than a threshold then the original software code will get restored. The *KeySplitWatermark* is evaluated on three different software samples and two different watermarks. Experimental results prove the robustness of the algorithm under attacks.

We propose *KeySplitWatermark*, a novel approach based on blind zero watermarking to protect software source code against cyber-attack. The algorithm is blind and adds watermark logically into the code using the inherent properties of code and provides a robust solution. The algorithm is made up of two constituents: embedding algorithm and extraction algorithm. The algorithm creates a key using a watermark and can retrieve the key of the software even after it gets attacked or tampered. In case software undergoes an attack and tampering is detected, the original code can be restored causing attack effects to get nullified. Specifically, this work makes the following contributions to the cybersecurity and software watermarking community in the following ways:

1) A novel approach based on watermarking to protect software code against the attack that does not alter the software code to embed watermark.
2) No assumption about software code, programming language or length is made.
3) A novel reactive approach to cyber-attacks to increase software resilience.
4) Cyber-attacks on software code can be detected and the original software code can be restored if tampering increases to a user-defined threshold.

The remainder of the paper is structured as follows: Section II provides an overview of the previous efforts done in the software watermarking domain and for the cybersecurity of software. The proposed embedding and extraction algorithms are stated in detail in Section III. Section IV presents experimental results under tampering attacks with three distinct samples and two watermarks. We evaluate the performance of the proposed approach on a total of nine attacked samples. Section V presents a comparative analysis with state-of-the-art works. Section VI concludes this paper along with the directions for the future works.

II. LITERATURE REVIEW

Software watermarking is an evolving research area and several competitive software companies are claiming to be creating secure software codes. A robust, efficient and resilient software watermarking solution can be a game-changer for the information security [6]–[9] and software development community. Cyber-attacks on software are very common these days as more and more software vulnerabilities are being disclosed. Digital watermarking can provide solutions that can be helpful for software protection, content authentication, integrity checking, and fingerprinting. Several guidelines and approaches do exist for secure software development [3].

Davis [5] provided an overview of existing processes, standards, life-cycle models, frameworks, and methodologies that support or could support secure software development. McGraw [10] suggested 7 touchpoint activity areas, connecting to software development artifacts to build secure software.

Vulnerability scanning approach [11] based on an automated pattern-matching tool has also been used to identify vulnerabilities in software code. OWASP [12] provides a standard Secure Software Development Life Cycle and helps developers to know what should be considered or best practices at each phase of a development Life Cycle and has recently suggested the top 10 security controls for web application developers. Figure 3 gives a brief overview of currently available software protection tools.

Many software watermarking methodologies have been proposed in the last several decades [12]–[14]. The existing approaches towards software security are proactive approaches where the focus is on developing secure software following all stages of secure software development life cycle.

Software watermarking has been done in several ways during the last two decades, which includes static and dynamic software watermarking techniques. These techniques are based on software code, FP tree, registration allocation, graph-based, dynamic path, and many others. Some of the major techniques are grouped as follows:

A. RE-ORDERING ALGORITHMS

Re-ordering algorithms are static software watermarking algorithms that use semantics-preserving transformations to place a watermark in a permutation of the existing code. Davidson and Myhrvold proposed the first block reordering algorithm in 1996 [13]. Later, Myles *et al.* evaluated the effectiveness of this algorithm on Java byte code using Sandmark [14]. Gong *et al.* proposed a watermarking method for Java that analyzes the format of the Java class file and then re-order the indexes to embed watermark [15].

B. REGISTER ALLOCATION ALGORITHMS

Register allocation is considered as constraint-based static software watermarking technique. Based on this concept, Qu and Potkonjak suggested a QP algorithm for watermarking the graph coloring problem through register allocation [16]. After this, Myles and Collberg implemented this algorithm in Sand Mark, named as QPS algorithm and performed its empirical evaluation [17]. Later, Zhu and Thomborson proposed a further improvement which they call the QPI algorithm [18].

C. SPREAD-SPECTRUM ALGORITHMS

These methods utilize ideas from spread spectrum radio communications. Cox *et al.* proposed the idea to insert watermark in spectral components of data [19]. Later, Stern *et al.* introduced a robust object watermarking scheme which was more resilient against collusion attacks [20].

D. OPAQUE PREDICATE ALGORITHMS

Collberg et. al proposed the idea of opaque constructs in 1998 [21]. Arboit et. al proposed two methods for watermarking Java programs that use opaque predicates [22]. Later, this method was assessed by Myles and Collberg who implemented both static and dynamic versions within the SandMark framework [23].

E. ABSTRACT INTERPRETATION ALGORITHMS

Abstract interpretation is a static analysis technique used for, among other things, the verification of software. Cousot and Cousot presented an abstract interpretation algorithm that embedded the watermark in values assigned to selected integer local variables at run time [24]. Preda and Pasqua proposed a semantic approach to software watermarking where they modeled the ability of the attacker to identify the signature in the framework of abstract interpretation as a completeness property [25].

F. DYNAMIC PATH AND GRAPH-BASED ALGORITHMS

Collberg *et al.* proposed a dynamic path algorithm that inserts a watermark in the runtime branch structure of a program [26]. Graph-based watermarking algorithms rely on the fact that graph-generating code is difficult to analyze. Collberg and Thomborson proposed the first dynamic graph-based software watermarking algorithm [27], [28].

G. CODE REPLACEMENT ALGORITHMS

First patented software watermarking efforts used the idea of code replacement; that is, the watermark value replaced the pre-decided part of code [29], [30]. Monden *et al.* have explored watermarking of java programs and proposed several techniques by swapping byte code within dummy methods (implemented as jmark) [31], [32].

Yu *et al.* proposed an algorithm for software protection in cloud [33]. Guang et. al then recently proposed an algorithm for the protection of software in the cloud by rigorous theoretic treatment [34]. Hayoma *et al.* proposed a dynamic software watermarking approach using return-oriented programming [35]. Owned *et al.* shed light on the importance of security against malware and hijacking techniques [36]–[38].

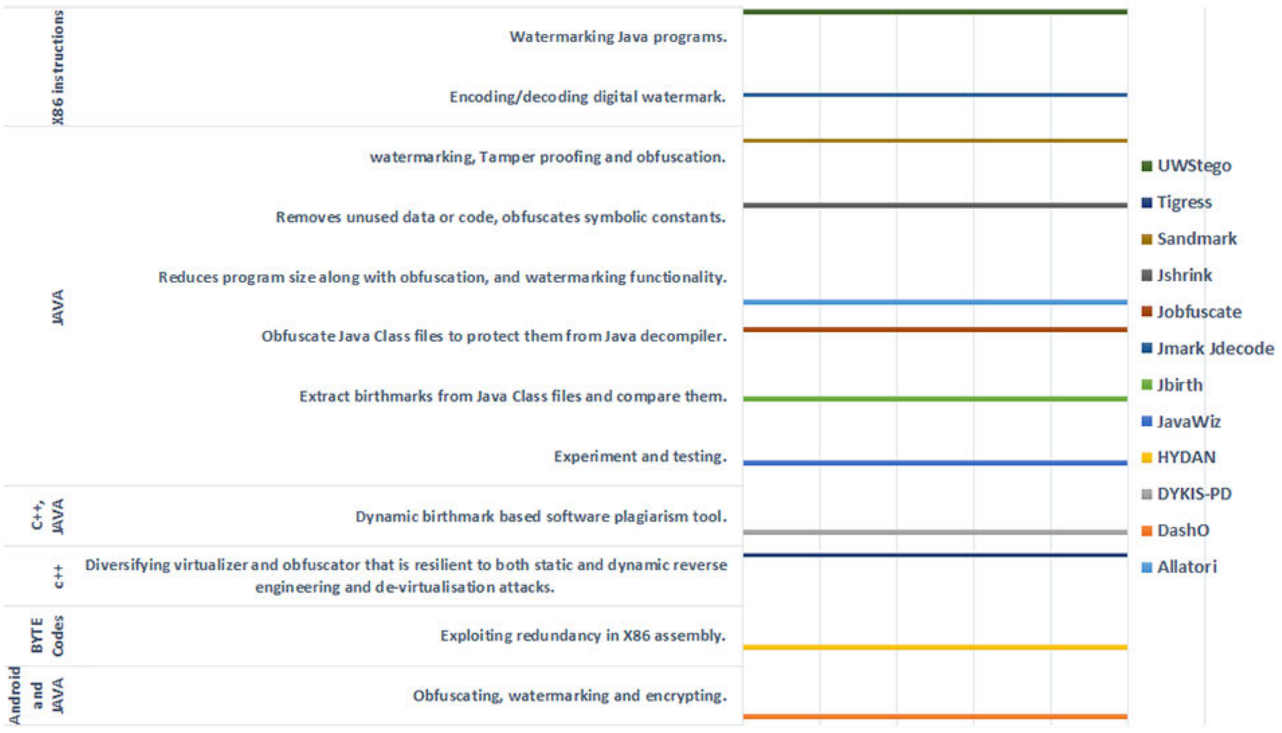

FIGURE 3. Tools for software watermarking used in literature.

Protecting software against cyber-attacks is crucial and software watermarking is one of the solutions which can be used to protect it after the attack on it happens and it can prevent damages. The existing approaches to software security are proactive and focus more on writing secure codes but none of the approaches focuses on making software resilient once the attack has already happened.

III. PROPOSED WORK

Protecting software against cyber-attacks is one of the most important concerns in the digital community. Watermarking which was initially used for copyright protection can now be used for software protection. The process of incorporation of a watermark into software that can uniquely identify the copyright owner of the software is called software watermarking. Software watermarking proves ownership but besides this, it also identifies tampering. If tampering increases to a certain threshold or follows a certain pattern, it may be a clear indicator for an upcoming cyber-attack. The upcoming cyber-attack can make that software a malware or a bot used for the next attack. In case tampering gets detected in real-time, the tampered version of the software can be replaced with the original version (registered with CA in the name of the copyright owner). In this way, an attacker won't be able to launch an attack. Many programming languages for software, its existence in executable form and dynamic interpretation and storage are some of the challenges and properties that need to be considered by any software watermarking technique. The integral requirements of a standard watermarking technique like robustness, imperceptibility, capacity, and security also need to be addressed.

We propose novel watermarking based algorithms for the protection of computer software against attacks. *KeySplit-Watermark* first analyzes software code to identify the keywords then make the partitions of the code on the basis of the selected keyword. The algorithm generates a unique key using the keywords and software code itself. If any copyright concern is raised in the future, this key can be used to demonstrate ownership. The embedding algorithm does not perform any tampering in software code to watermark it and extraction algorithms do not require watermark as input which makes it blind.

Figure 4 explains the *KeySplitWatermark* where embedding algorithm takes the following inputs:

1) Original code: Original software code which is to be watermarked.
2) Cipher: A digital value to be used in the key generation process.
3) Watermark: A group of ASCII characters.

The embedding algorithm generates the owner key as an output. That key is recorded with the CA and then further used to extract watermark (if needed). The extraction algorithm takes the following inputs:

1) Attacked code file: A software code file that is tampered with and or used illegally as copyright infringement.

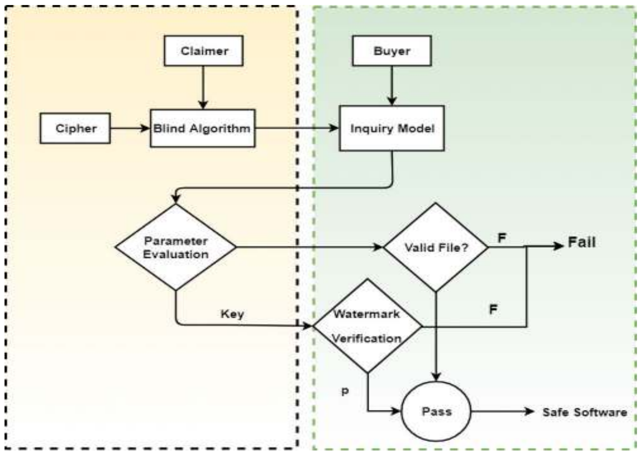

FIGURE 4. Block diagram of proposed approach.

2) Owner key: It is acquired from the certification authority to identify the original owner.

A trusted Certificate Authority (CA) is a requirement for this algorithm that registers the contents in the name of the copyright owner. Whenever an attack is suspected, this trusted third party performs watermark extraction and in case tampering is detected, it provides the original software code for restoration. The tampered code gets replaced with original code making the attacker's actions null and void. The watermarking algorithm is made up of two constituents; watermark embedding and watermark extraction. Watermark embedding is performed by the original owner of the software and extraction is done later by a trusted third party.

The extraction algorithm takes two inputs: 1) attacked code and 2) owner key, and then extracts the watermark. This watermark later proves the identity of the original owner and then to restore the original code.

A. EMBEDDING ALGORITHM

The embedding algorithm inserts the watermark in the software code/program and generates an owner key utilizing keywords of the programming language in the program. The detailed watermark embedding algorithm is stated as follows:

In this algorithm, software code is first preprocessed to identify the most occurring ten characters and the most occurring five keywords. It is then partitioned based on the user-selected keyword. After that, the maximum occurring alphabetical character is identified from each partition to populate the MOC list. This MOC list is used to create the owner key based on the given watermark. Each letter of the watermark is compared with each letter in the MOC list if it is matched then the key is populated with digit 0 (indicating direct method) and partition number (PN). If the letter of the watermark is not found in the MOC list then it is populated by using shift ciphering method (SC), but first with digit 1 indicating indicates indirect method. The keyword is then combined with the key to generating the owner key (OK). The original watermark (W) and owner key (OK) are registered

Algorithm 1 Working of Embedding Algorithm (z_{26}) Represents the 26 Alphabets (a-Z)

```
1  Input C
2  Preprocess C
3  Count occurrences of all characters
4  Count occurrences of all keywords
5  Display top 10 character list and input W
6  Display top 5 keywords and input KW
7  Partition C based on KW
8  Identifying MOC from each partition and populate
   MOC list
9  For each letter of W, repeat step 10
10 if $w_j \epsilon MOC$ then
11     key[i]=0;
12     where j=1,2,...wl keyi+1=PN(MOC)
13 else if $w_j \notin MOC$ then
14     key[i]=0; where j=1,2,...wl
15     keyi+1=PN(MOC)
16 else
17     key[1+1]=$(w_j + k) Mod 26$
18     where k is in $z_{26}$
19 OK = concatenate (KW, Key)
20 Output OK
```

W	Watermark	*KW*	Keyword
SC	Shift Cipher	*OK*	Owner Key
C	Software source code	*PN*	Partition number
MOC	Maximum Occurring Character		

with a certification authority with original code, date and time. This code would be used to replace the tampered code at a later stage if the need arises.

B. EXTRACTION ALGORITHM

The extraction algorithm extracts the watermark from the software code when needed. It uses the watermark key previously generated by the embedding algorithm as input and extracts watermark from the attacked software code. This algorithm is kept with the trusted third party i.e. Certification Authority.

The extraction algorithm is as follows:

In this algorithm, the attacked code (C_a) is partitioned using the keyword (KW) obtained from the owner key (OK). After this, the maximum occurring character (MOC) from each partition is identified and the MOC list is populated. The contents of the watermark key (WK) are later used to attain watermark from the source code. The extracted watermark can then be matched with the original watermark previously registered with CA to prove ownership by using any pattern matching metric (e.g. similarity index). In case matching increases a user-defined threshold or in case a certain tampering pattern is found, the original software code replaces the tampered code.

Algorithm 2 Working of Extraction Algorithm

```
1  Input $C_a$
2  Preprocess $C_a$
3  Partition C based on KW (Obtained from OK)
4  Identify MOC from each partition and make MOC list
5  $L_1$ = length (KW), keyindex = $L_1 + 1$, $L_2$=length(W)
6  while $keyindex < L_2$ do
7      if $OKi^{++} = 0$ then
8          $W_e$(I)= MOC (PN)
9      else
10         $W_e(I) = ReverseSC$
11     Increment I
12 Output $W_e$
```

C_a	Attacked code	KW	Keyword
W	Watermark	W_e	Extracted Watermak
OK	Owner Key	SC	Shift Cipher
MOC	Maximum Occurring Character		

IV. EXPERIMENTAL RESULTS

To evaluate the performance of the proposed *KeySplitWatermark*, we used three different samples of software source codes and we name these as S1, S2, and S3. These samples are obtained from three independent sources of code written in the C++ programming language, a bus reservation system [39], a hospital management system [40] and a student information system [41]. These code samples include classes, public member functions, iteration, and decision structures. The *KeySplitWatermark* does not dependent on any specific programming language and can be executed in the same way on codes written in other programming languages.

Two different watermark samples (WM1 and WM2) are used of 370 and 592 characters respectively. Details about three software code samples (S1, S2, and S3) and two watermark samples (WM1 and WM2) obtained from [42] and [43] are mentioned in Table 6. The chosen samples vary in several lines of words, lines of code (LOCs) and several special characters.

Possible attacks on software source code are termed as tampering which means random insertion and deletion of words and lines; to and from the software source code. In code obfuscation attacks, software codes are modified to serve the desired purpose whether it be a malicious activity or copyright violation or any other. Tampering can be made at a single location in code to make it do something different which is termed as localized tampering or it is done at multiple different points in code which is named as dispersed tampering. The location and volume of tampering in software code cannot be anticipated in advance by software owner as he/she may not know the intention of the attacker. Generally, attackers make combined insertion and deletion attack to insert their piece of malicious code. Therefore, the *KeySplitWatermark* is evaluated against such type of attacks.

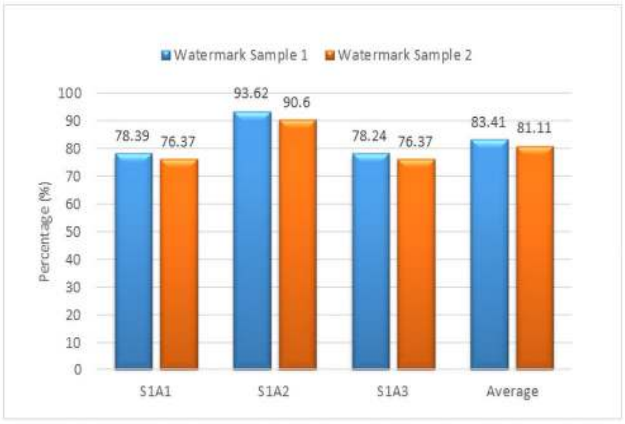

FIGURE 5. Accuracy of the retrieved watermark samples with both watermarks.

The original code samples S1, S2 and S3 are given to three different individuals (software programmers) to perform tampering attacks independently. Three attack samples of each sample are obtained. Details of attacked code samples can be seen in Tables 2, 3 and 4. Table 2 shows that for S1, attackers reduced lines of code and number of words significantly in first and third attacked samples (S1A1 and S1A3).

Code sample S2 is also attacked significantly which resulted in the reduction of lines of codes (LOCs) for all three attack samples, particularly in the first sample (S2A1) more than 80 lines of codes are removed and up to 500 characters are removed (Table 3).

Table 4 shows the attack impact on code sample 3 where more than 90 lines of codes got reduced for attacked sample 3 (S3A3) representing significant modification on the original code sample.

We conduct experiments to assess the performance of the *KeySplitWatermark* on all the attacked samples. The extracted watermarks are compared with the original watermarks generated by the embedding algorithm using original code samples.

First, we evaluate the performance of the *KeySplitWatermark* on code sample 1 (S1) using both watermarks WM1 and WM2 on all three attacked code samples (S1A1, S1A2, and S1A3). Figure 5 shows the accuracy of the retrieved watermark samples with both watermarks. It can be observed that watermark accuracy is more than 76% on both watermarks for all samples and above 81% on average. This is considered sufficient to identify the original copyright owner of the software code. Once the original copyright owner gets identified, the original software code replaces tampered code and cancels the attacker's efforts of either performing attack on a user's system or making it perform the network-based attack. Hence, the performance/functionality of software remains unaltered and exploit codes make no use for attackers shown in Figure 5.

Next, a similar experiment is performed for sample 2 (S2). Figure 6 shows the accuracy of the obtained watermark on attacked samples (S2A1, S2A2, and S2A3).

TABLE 1. Detail of original code files and watermark samples.

Sample	Total lines of code	Total words	Total alphabets	Total special characters	Total character	Total keywords	Total digits
S1	404	365	1721	993	2791	12	77
S2	587	1059	4178	1016	5271	12	77
S3	335	459	1732	725	2497	11	40
WM1	6	67	361	9	370	0	0
WM2	8	114	563	18	592	0	11

TABLE 2. Details of three attacked samples (S1A1, S1A2, AND S1A3) obtained after attack on original code sample S1.

Attributes	Original sample 1 (S1)	Attacked sample 1 (S1A1)	Attacked sample 2 (S1A2)	Attacked sample 3 (S1A3)
Total lines of code	404	310	344	358
Total words	365	290	364	331
Total Alphabets	1721	1443	1804	1626
Total special characters	993	794	993	892
Total characters	2791	2297	2874	2586
Total Digits	77	60	77	68

TABLE 3. Details of three attacked samples (S2A1, S2A2, AND S2A3) obtained after attack on original code sample S2.

Attributes	Original Sample 3 (S3)	Attacked sample 1 (S3A1)	Attacked sample 2 (S3A2)	Attacked sample 3 (S3A3)
Total lines of code	335	292	246	238
Total words	459	453	486	279
Total alphabets	1732	1691	1759	1069
Total special characters	725	725	735	433
Total Characters	2497	2456	2532	1530
Total digits	40	40	38	28

TABLE 4. Details of three attacked samples (S3A1, S3A2, AND S3A3) obtained after attack on original code sample S3.

Attributes	Original Sample 2 (S2)	Attacked sample 1 (S2A1)	Attacked sample 2 (S2A2)	Attacked sample 3 (S2A3)
Total lines of code	587	499	527	501
Total words	1059	928	924	903
Total alphabets	4178	3639	3634	3588
Total special characters	1016	903	925	853
Total characters	5271	4612	4624	4507
Total digits	77	70	65	67

TABLE 5. Comparative table of the State-of-art-work. Ke GT- Graph Theoretic, PB - Path/branch, MM - Mathematical Model, EB - Encoded Binary.

Authors	Techniques	Approach	Key findings	Limitations
Mpanti et al. [39]	Subtractive	GT	Reducible permutation flow-graphs	Constraints of bytecode
Chen et al. [40]	Distortive	GT	Improved the low efficiency of PPCT dynamic graph encoding	Constraints of bytecode
Z. Chen et al. [41]	Distortive	PB	Construct a hidden execution path	Varying Path Length
Nazir et al. [42]	Subtractive	MM	Mathematical approach to deal with vagueness and uncertainty	Static approach
Chen et al. [43]	Distortive	PB	Neural network for temper detection	Varying Path Length
Wang et al. [44]	Distortive	EB	Exception handling based of dynamic software	Complexity

Average watermark accuracy is above 70% using both watermarks. S2A1 is a significantly attacked sample. Accuracy of extracted watermarks on this sample is 76.11% and 71.58% for WM1 and WM2 respectively which shows the robustness of *KeySplitWatermark*.

Then, we test our *KeySplitWatermark*s on the third code sample (S3). Figure 7 shows the accuracy of the obtained watermark on attacked samples (S3A1, S3A2, and S3A3). The average watermark accuracy for both watermarks remained greater than 90%. S3A3 sample is obtained after massive tampering (reduction in 90 LoCs) but we are still able to achieve 81.16 and 79.75% accuracy on that sample.

Experimental results show that the algorithm poses good resistance against tampering attacks of different types and volume as even in the worst case (S2A1) 69.56% watermark is successfully retrieved which is sufficient to identify the original software owner. The results prove that the proposed system is robust, practical and secure against random tampering attacks on all nine samples. Watermarks survived combined insertion and deletion attack which are both localized and dispersed. It provides better security since the attacker won't be aware of the existence of such an anti-malware system. Also, the algorithm has linear complexity which makes it computationally efficient and obtains better accuracy. Getting user input with a watermark to produce key in embedding algorithm makes it more robust as there are least chances of having the same key for two different software codes even with the same watermark.

Using Zero-watermarking of static software code and particularly to prevent against cyber-attack is a novel idea and no such zero software watermarking algorithms are available to make a comparison of our proposed approach. However, it is hereby claimed that none of the previous static software watermarking gives above 80% watermark accuracy on

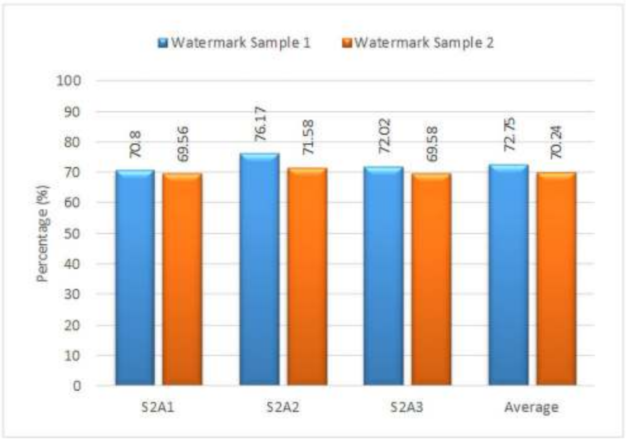

FIGURE 6. Accuracy of obtained watermark samples on three attacked samples (S2A1, S2A2, and S2A3) obtained after the attack on original code sample S2.

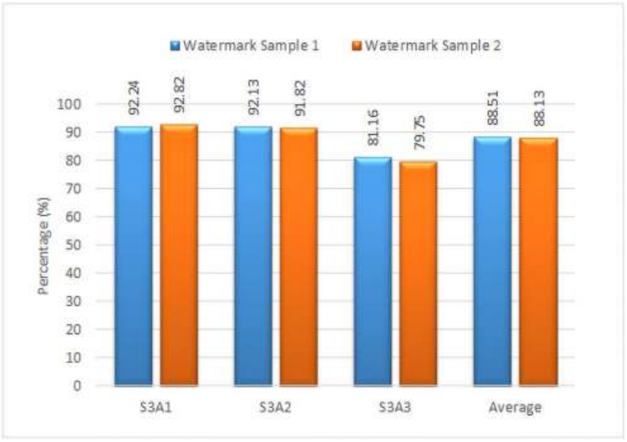

FIGURE 7. Accuracy of obtained watermark samples on three attacked samples (S3A1, S3A2 and S3A3) obtained after attack on original code sample S3.

average and to our best knowledge, watermarking has not been used as a way to prevent cyber-attacks in past.

The *KeySplitWatermark* can be applied to protect software source codes written in any programming language and of finite length against attacks. The proposed approach can be deployed as an anti-malware system and can make the attacker's efforts null and void. Also, we can detect and observe the tampered part of code to identify attackers' intentions.

V. COMPARATIVE ANALYSIS

This section compares the performance of the *KeySplitWatermark* with the most recent relevant work on software watermarking Experiments are carried out on this two programs: CompressDemo and CryptoEncryption used in previous research work by [40], to facilitate comparison and used watermarks for 128, 256, 512, and 1024-bit random binary sequences. The experimental setup is kept the same as in [40] and the experiments are conducted in a system environment with Intel Core I5 CPU, 4GB of RAM, and Windows 10 operating system.

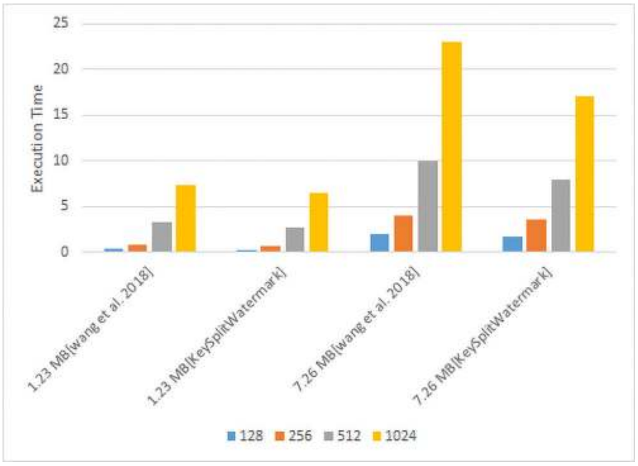

FIGURE 8. Execution time of *KeySplitWatermark* as compared with [40] for CompressDemo program for 1.23, and 7.26 MB of data with varied size watermarks.

To evaluate the watermarked program execution time objectively, two different inputs are selected for each program. To reduce the interference of the operating system, memory, and other environments on program execution time, we run the programs 20 times with each input and calculated the average time. The comparative results of experiments performed on the CompressDemo program with 1.23 MB files as input and varied sizes of watermarks are given in Table 6. The *KeySplitWatermark* is zero watermarking algorithm, hence it does not utilize exceptions, so no binary encoding for exceptions is needed. With our algorithm, there is no increase in file size and execution time.

The comparative results of experiments performed on CryptoEncryption program with 31KB file as input and varied size of watermarks are given in table 2. The reduction in execution time can be observed in *KeySplitWatermark* due to zero watermarking approach and no additional computation needed for exception handling.

The reduction in execution time of the watermarked CryptoEncryption and CompressDemo for [40] and for *KeySplitWatermark* with different input sizes are shown in Figures 8 and 9, respectively. Figure 8 shows the reduction in execution time (ms) for input files of size 1.23 MB and 7.26 MB for the *KeySplitWatermark* as compared with [40] after the watermarks of 128, 256, 512 and 1024 bits are embedded in CompressDemo program. The *KeySplitWatermark* has reduced execution time with both input files even when watermark size is 1024 bits.

Figure 9 shows the reduction in execution time (ms) for input files of size 3 and 48 KB for *KeySplitWatermark* as compared with [40] after the watermarks of 128, 256, 512 and 1024 bits are embedded in CryptoEncryption program. The *KeySplitWatermark* reduce the execution time with both input files for watermarks of varied sizes.

The execution time of the *KeySplitWatermark* is not dependent on watermark length. As the input size increases, the watermarked program execution time increases too but

TABLE 6. Comparative results for increase in the size of the watermarked code and in execution time for CompressDemo with 1.23 MB file.

Watermark length (bits)	Increase in program size (KB) [40]	Increase in program size *KeySplitWatermark*	Execution time (ms) [40]	Execution time (ms) *KeySplitWatermark*
128	15	**0**	0.36	**0.28**
256	33	**0**	0.90	**0.75**
512	67	**0**	3.25	**2.78**
1024	130	**0**	7.30	**6.54**

TABLE 7. Comparative results for increase in the size of the watermarked code and in execution time for CrptoEncryption with 31KB file.

Watermark length (bits)	Increase in program size (KB) [40]	Increase in program size *KeySplitWatermark*	Execution time (ms) [40]	Execution time (ms) *KeySplitWatermark*
128	18	**0**	23	**18**
256	34	**0**	40	**32**
512	67	**0**	45	**39**
1024	130	**0**	123	**105**

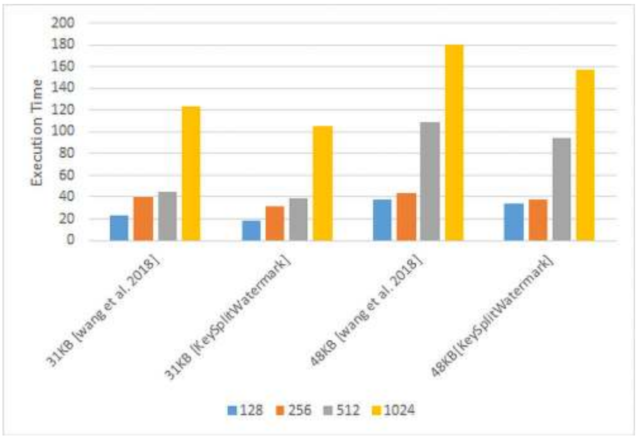

FIGURE 9. Execution time of *KeySplitWatermark* as compared with [40] for CryptoEncryption program for with different inputs and varied size watermarks.

that is a natural increase with no overhead of exception handling as in [40]. The algorithm proposed by Wang *et al.* [40] shows degraded performance on programs with a large number of loops, since if the watermarks are embedded in these loops, large quantities of exception handling get executed, resulting in a significant increase in program execution time. In *KeySplitWatermark*, an increase in the number of loops does not make any effect on program execution time even with large inputs.

To further evaluate the robustness of the *KeySplitWatermark*, we use attack tools ASProtect, Upx, and Aspack to attack the watermarked program and verify the correctness of the extracted watermark. The experimental results are shown in Table 8. The watermark can be extracted correctly after attacks of encryption, shelling, and compression of the watermarked programs. The original semantics of the program are still maintained, although different attacks are conducted.

The existing watermarking algorithms for static as well as dynamic watermarking embed watermark in program files which increase file size and execution time. In our proposed method, we make no changes in program file, rather generate key using the structure of file thus make no increase in program size. The *KeySplitWatermark* also extracts 100% watermark since the compression, encryption and shelling does not change program code. Wang *et al.* [40] also extracted 100% watermark but at the cost of increased execution time and increased file size. In our proposed approach, even if program code gets tampered and 80% watermark sequence gets extracted, it would be sufficient to claim copyright ownership. In *KeySplitWatermark*, where watermarking embedding is done once only and extraction is done only if any copyright issues arises. There is no need to increase program size and enhance execution time of program for watermark embedding.

TABLE 8. Attacks and results.

Tool	Attack Mode	Extraction [40]	Extraction *KeySplitWatermark*
ASProtect	Encrypts program	100%	100%
Upx	Conducts code compression	100%	100%
Aspack	Used to shell the program	100%	100%

VI. CONCLUSION

In this work, we proposed *KeySplitWatermark*, a novel zero watermarking approach to protect software code against cyber-attacks. The algorithm is blind and adds watermark logically into the code using the inherent properties of code and provides a robust solution. The source code and the user-provided watermark are used to produce a personalized key that gets registered with the Certification Authority (CA) and is used later by the extraction algorithm to identify the original owner and to restore the original software code. Our *KeySplitWatermark* logically embeds watermark, the presence of watermark is known by the original owner and CA only. It is not possible to destroy the watermark without altering the code significantly and if any change occurs in code, the original code gets restored. The performance of the algorithm was evaluated for tampering attacks made on three different code samples and the results prove that the

KeySplitWatermark is robust, secure and efficient with minimal computational requirements. We also compared the proposed algorithm with the relevant work and it outperformed in terms of execution time, capacity and size. In this work, we have used two watermarks and limited samples written in two programming languages. In the future, this work can be extended and evaluated for application-specific software codes written in other programming languages with different set and number of keywords.

REFERENCES

[1] A. R. Javed, M. O. Beg, M. Asim, T. Baker, and A. H. Al-Bayatti, ''AlphaLogger: Detecting motion-based side-channel attack using smartphone keystrokes,'' *J. Ambient Intell. Humanized Comput.*, pp. 1–14, Feb. 2020.

[2] A. K. Abdulrahman and S. Ozturk, ''A novel hybrid DCT and DWT based robust watermarking algorithm for color images,'' *Multimedia Tools Appl.*, vol. 78, no. 12, pp. 17027–17049, Jun. 2019.

[3] W. Hu, R.-G. Zhou, J. Luo, and B. Liu, ''LSBs-based quantum color images watermarking algorithm in edge region,'' *Quantum Inf. Process.*, vol. 18, no. 1, p. 16, Jan. 2019.

[4] Z. Jalil and A. M. Mirza, ''An invisible text watermarking algorithm using image watermark,'' in *Innovations in Computing Sciences and Software Engineering*. Dordrecht, The Netherlands: Springer, 2010, pp. 147–152.

[5] N. Davis, ''Secure software development life cycle processes: A technology scouting report,'' Carnegie-Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. ADA447047, 2005.

[6] M. E. Kumar, G. T. Reddy, K. Sudheer, M. Reddy, R. Kaluri, D. S. Rajput, and K. Lakshmanna, ''Vehicle theft identification and intimation using gsm & iot,'' in *Proc. Mater. Sci. Eng. Conf.*, vol. 4, 2017, Art. no. 042062.

[7] G. T. Reddy, R. Kaluri, P. K. Reddy, K. Lakshmanna, S. Koppu, and D. S. Rajput, ''A novel approach for home surveillance system using IoT adaptive security,'' in *Proc. Int. Conf. Sustain. Comput. Sci., Technol. Manage. (SUSCOM)*, vol. 3. Rajasthan, India: Amity Univ. Rajasthan, Feb. 2019, pp. 1616–1625, doi: 10.2139/ssrn.3356525.

[8] G. T. Reddy, K. Sudheer, K. Rajesh, and K. Lakshmanna, ''Employing data mining on highly secured private clouds for implementing a security-asa-service framework,'' *J. Theor. Appl. Inf. Technol.*, vol. 59, no. 2, pp. 317–326, 2014.

[9] R. Raghavan, J. K. Singh, T. G. Reddy, K. Sudheer, P. Venkatesh, and S. O. Olabiyisi, ''A case study: Home environment monitoring system using Internet of Things,'' *Int. J. Mech. Eng. Technol.*, vol. 8, no. 11, pp. 173–180, 2017.

[10] J. Epstein, S. Matsumoto, and G. McGraw, ''Software security and SOA: Danger, will robinson!'' *IEEE Secur. Privacy Mag.*, vol. 4, no. 1, pp. 80–83, Jan. 2006.

[11] A. Al-Ghamdi, ''A survey on software security testing techniques,'' *Int. J. Comput. Sci. Telecommun.*, vol. 4, pp. 14–18, Apr. 2013.

[12] G. S. Leite and A. B. Albuquerque, ''The importance of safe coding practices and possible impacts on the lack of their application,'' in *Proc. Comput. Sci. On-line Conf.* Cham, Switzerland: Springer, 2019, pp. 214–224.

[13] R. I. Davidson and N. Myhrvold, ''Method and system for generating and auditing a signature for a computer program,'' U.S. Patent 5 559 884, Sep. 24, 1996.

[14] G. Myles, C. Collberg, Z. Heidepriem, and A. Navabi, ''The evaluation of two software watermarking algorithms,'' *Softw., Pract. Exper.*, vol. 35, no. 10, pp. 923–938, Aug. 2005.

[15] D. Gong, F. Liu, B. Lu, P. Wang, and L. Ding, ''Hiding informationin in java class file,'' in *Proc. Int. Symp. Comput. Sci. Comput. Technol.*, vol. 2, 2008, pp. 160–164.

[16] G. Qu and K. Potkonjak, ''Analysis of watermarking techniques for graph coloring problem,'' in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design. Dig. Tech. Papers*, Nov. 1998, pp. 190–193.

[17] G. Myles and C. Collberg, ''Software watermarking through register allocation: Implementation, analysis, and attacks,'' in *Proc. Int. Conf. Inf. Secur. Cryptol.* Berlin, Germany: Springer, 2003, pp. 274–293.

[18] W. Zhu and C. Thomborson, ''Algorithms to watermark software through register allocation,'' in *Proc. Int. Conf. Digit. Rights Manage.* Berlin, Germany: Springer, 2005, pp. 180–191.

[19] I. J. Cox, J. Kilian, T. Leighton, and T. Shamoon, ''A secure, robust watermark for multimedia,'' in *Proc. Int. Workshop Inf. Hiding*. Berlin, Germany: Springer, 1996, pp. 185–206.

[20] J. P. Stern, G. Hachez, F. Koeune, and J.-J. Quisquater, ''Robust object watermarking: Application to code,'' in *Proc. Int. Workshop Inf. Hiding*. Berlin, Germany: Springer, 1999, pp. 368–378.

[21] C. Collberg, C. Thomborson, and D. Low, ''Manufacturing cheap, resilient, and stealthy opaque constructs,'' in *Proc. 25th ACM SIGPLAN-SIGACT Symp. Princ. Program. Lang. (POPL)*, 1998, pp. 184–196.

[22] G. Arboit, ''A method for watermarking java programs via opaque predicates,'' in *Proc. 5th Int. Conf. Electron. Commerce Res. (ICECR)*, 2002, pp. 102–110.

[23] G. Myles and C. Collberg, ''Software watermarking via opaque predicates: Implementation, analysis, and attacks,'' *Electron. Commerce Res.*, vol. 6, no. 2, pp. 155–171, Apr. 2006.

[24] P. Cousot and R. Cousot, ''An abstract interpretation-based framework for software watermarking,'' *ACM SIGPLAN Notices*, vol. 39, no. 1, pp. 173–185, Jan. 2004.

[25] M. Dalla Preda and M. Pasqua, ''Software watermarking: A semantics-based approach,'' *Electron. Notes Theor. Comput. Sci.*, vol. 331, pp. 71–85, Mar. 2017.

[26] C. Collberg, E. Carter, S. Debray, A. Huntwork, J. Kececioglu, C. Linn, and M. Stepp, ''Dynamic path-based software watermarking,'' in *Proc. ACM SIGPLAN Conf. Program. Lang. design Implement.*, 2004, pp. 107–118.

[27] C. Collberg and C. Thomborson, ''Software watermarking: Models and dynamic embeddings,'' in *Proc. 26th ACM SIGPLAN-SIGACT Symp. Princ. Program. Lang. (POPL)*, 1999, pp. 311–324.

[28] C. S. Collberg and C. Thomborson, ''Watermarking, tamper-proofing, and obfuscation–tools for software protection,'' *IEEE Trans. Softw. Eng.*, vol. 28, no. 8, pp. 735–746, Aug. 2002.

[29] K. Holmes, ''Computer software protection,'' U.S. Patent 5 287 407, Feb. 15, 1994.

[30] P. R. Samson, ''Apparatus and method for serializing and validating copies of computer software,'' U.S. Patent 5 287 408, Feb. 15, 1994.

[31] J. Hamilton and S. Danicic, ''A survey of static software watermarking,'' in *Proc. World Congr. Internet Secur. (WorldCIS)*, Feb. 2011, pp. 100–107.

[32] B. B. Madan, M. Banik, and D. Bein, ''Securing unmanned autonomous systems from cyber threats,'' *J. Defense Model. Simul., Appl., Methodol., Technol.*, vol. 16, no. 2, pp. 119–136, Apr. 2019.

[33] A. Dey, S. Bhattacharya, and N. Chaki, ''Software watermarking: Progress and challenges,'' *INAE Lett.*, vol. 4, no. 1, pp. 65–75, Mar. 2019.

[34] S. Guang, F. Xiaoping, F. Sha, S. Yingjie, and L. Huifang, ''Software watermarking in the cloud: Analysis and rigorous theoretic treatment,'' *J. Softw. Eng.*, vol. 9, no. 2, pp. 410–418, Feb. 2015.

[35] H. Ma, K. Lu, X. Ma, H. Zhang, C. Jia, and D. Gao, ''Software watermarking using return-oriented programming,'' in *Proc. 10th ACM Symp. Inf., Comput. Commun. Secur. (ASIA CCS)*, 2015, pp. 369–380.

[36] C. Iwendi, M. Uddin, J. A. Ansere, P. Nkurunziza, J. H. Anajemba, and A. K. Bashir, ''On detection of Sybil attack in large-scale VANETs using spider-monkey technique,'' *IEEE Access*, vol. 6, pp. 47258–47267, 2018.

[37] C. Iwendi, A. Allen, and K. Offor, ''Smart security implementation for wireless sensor network nodes,'' *J. Wireless Sensor Netw.*, vol. 1, no. 1, pp. 1–13, 2015.

[38] C. O. Iwendi and A. R. Allen, ''Cia security management for wireless sensor network nodes,'' in *Proc. 12th Annu. PostGraduate Symp. Converg. Telecommun., Netw. Broadcast.* Liverpool, U.K.: Liverpool John Moores Univ., 2011, pp. 123–128.

[39] K. Lu, S. Xiong, and D. Gao, ''RopSteg: Program steganography with return oriented programming,'' in *Proc. 4th ACM Conf. Data Appl. Secur. Privacy (CODASPY)*, 2014, pp. 265–272.

[40] Y. Wang, D. Gong, B. Lu, F. Xiang, and F. Liu, ''Exception handling-based dynamic software watermarking,'' *IEEE Access*, vol. 6, pp. 8882–8889, 2018.

[41] J. Wang, P. Xie, Y. Wang, and Z. Rong, ''A survey of return-oriented programming attack, defense and its benign use,'' in *Proc. 13th Asia Joint Conf. Inf. Secur. (AsiaJCIS)*, Aug. 2018, pp. 83–88.

[42] H. P. Joshi, A. Dhanasekaran, and R. Dutta, ''Impact of software obfuscation on susceptibility to return-oriented programming attacks,'' in *Proc. 36th IEEE Sarnoff Symp.*, Sep. 2015, pp. 161–166.

[43] A. Alrehily and V. Thayananthan, ''Software watermarking based on return-oriented programming for computer security,'' *Int. J. Comput. Appl.*, vol. 166, no. 8, pp. 21–28, 2017.

CELESTINE IWENDI (Senior Member, IEEE) received the master's degree in communication hardware and microsystem engineering from Uppsala University, Sweden, in 2008, and the Ph.D. degree in electronics from the University of Aberdeen, U.K., in 2013. He is an Associate Professor from Sweden. He ranked under 100 in the world university ranking. He is a highly motivated researcher with a *Wireless Sensor Network Security* book and over 100 publications. He is currently a Senior Lecturer with the Department of Electronics BCC, Central South University of Forestry and Technology, China. He has strong teaching emphasis on communication, hands-on experience, willing-to-learn and 18 years of technical expertise. He currently teaches engineering team project, circuit theory, data networks, and distributed systems, and control systems. He has developed operational, maintenance, and testing procedures for electronic products, components, equipment, and systems; provided technical support and instruction to staff and customers. He is a wireless sensor network Chief Evangelist, a Researcher, and a Designer. He has been a Board Member of the IEEE Sweden Section, since 2017, and a Fellow of The Higher Education Academy, U.K., to add to his teaching and professional experiences. He is an Editor of *International Journal of Engineering and Allied Disciplines*, in 2015, a Newsletter Editor of the IEEE Sweden Section, from 2016 to 2018, the Editor-in-Chief of *Wireless Sensor Network Magazine*, in 2009, a Committee Member of International Advisory Panel, International Conference on Marine, Ocean and Environmental Sciences and Technologies (MAROCENET), from 2014 to 2016, the Editor-in-Chief of *Journal of Wireless Sensor Network*, in 2009, and an Advisory Board member of *International Journal of Innovative Computer Science and Engineering* (IJICSE), in 2013. He is the Co-Chair of the special session on Wireless Sensor Networks.

ZUNERA JALIL (Member, IEEE) received the B.Sc. degree from Punjab University, Lahore, Pakistan, in 1999, and the M.S. degree in computer science and the Ph.D. degree in computer science with information security specialization from the FAST National University of Computer and Emerging Sciences, Islamabad, Pakistan, in 2007 and 2010, respectively. She is currently an Assistant Professor with the Department of Cyber Security and a Researcher with the National Cybercrimes and Forensics Laboratory, Air University, Islamabad. Her research interest includes but is not limited to computer forensics, intelligent systems, and data privacy protection.

ABDUL REHMAN JAVED received the master's degree in computer science from the FAST-National University of Computer and Emerging Sciences, Islamabad, Pakistan. He is currently a Research Assistant with the National Center for Cyber Security and a Visiting Lecturer with the Department of Computer Science, Air University, Islamabad. His research interests include but are not limited to mobile and ubiquitous computing, data analysis, knowledge discovery, data mining, natural language processing, smart homes, and their applications in human activity analysis, human motion analysis, and e-health. He aims to contribute to interdisciplinary research of computer science and human-related disciplines.

THIPPA REDDY G. received the B.Tech. degree in computer science and engineering from Nagarjuna University, Andhra Pradesh, India, the M.Eng. degree in computer science and engineering from Anna University, Chennai, India, and the Ph.D. degree from the Vellore Institute of Technology, Vellore, India. He is currently working as an Assistant Professor (Senior) with the School of Information Technology and Engineering, VIT, Vellore, India. He has 14 years of experience in teaching. He produced more than 25 international/national publications. His current research interests include machine learning, deep learning, computer vision, and big data analytics, Blockchain.

RAJESH KALURI received the B.Tech. degree in CSE from JNTU, Hyderabad, the M.Tech. degree in CSE from ANU, Guntur, India, and the Ph.D. degree in computer vision from VIT University, India. He is having more than ten years of teaching experience. He was a Visiting Professor with the Guangdong University of Technology, China, in 2015 and 2016. He is currently working as a Senior Assistant Professor with the School of Information Technology and Engineering, VIT University, India. He has published research articles in various reputed international journals. His current research is in the areas of computer vision, human-computer interaction, and blockchain.

GAUTAM SRIVASTAVA (Senior Member, IEEE) received the B.Sc. degree from Briar Cliff University, USA, in 2004, and the M.Sc. and Ph.D. degrees from the University of Victoria, Victoria, BC, Canada, in 2006 and 2011, respectively. He then taught for three years at the Department of Computer Science, University of Victoria, where he was regarded as one of the top undergraduate professors in the computer science course instruction. In 2014, he joined a tenure-track position at Brandon University, Brandon, MB, Canada, where he is currently active in various professional and scholarly activities. He was promoted to an Associate Professor, in January 2018. He, as he is popularly known, is active in research in the field of data mining and big data. In his eight-year academic career, he has published a total of 60 articles in high impact conferences in many countries and high-status journals (SCI and SCIE) and has also delivered guest lectures on big data, cloud computing, Internet of Things, and cryptography at many Taiwanese and Czech universities. He currently has active research projects with other academics in Taiwan, Singapore, Canada, Czech Republic, Poland, and USA. He is constantly looking for collaboration opportunities with foreign professors and students. He received the Best Oral Presenter Award in FSDM 2017 which was held at the National Dong Hwa University (NDHU), Hualien County, Shoufeng, Taiwan, in November 2017. He is an Editor of several international scientific research journals.

OHYUN JO (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), in 2005, 2007, and 2011, respectively. From April 2011 to February 2016, he was with Samsung Electronics in charge of research and development for future wireless communication systems, applications, and services. From March 2016 to July 2017, he was a Senior Researcher with the Electronics and Telecommunications Research Institute (ETRI) and from August 2017 to February 2018, he was an Assistant Professor with the Department of Electrical Engineering. He is currently an Assistant Professor with the Department of Computer Science, Chungbuk National University. He has authored or coauthored more than 40 articles and holds more than 150 registered and filed patents. His research interests include millimeter-wave communications, next-generation WLAN/WPAN systems, 5G mobile communication systems, military communications, Internet of Things, future wireless solutions/applications/services, machine learning, and embedded communications ASIC design. During his appointment at Samsung, he was a recipient of numerous recognitions, including Gold Prize in Samsung Annual Award, the Most Creative Researcher of the Year Award, the Best Mentoring Award, Major R & D Achievement Award, and the Best Improvement of Organization Culture Award.