Journal of Information Assurance and Security.
ISSN 1554-1010 Volume 7 (2012) pp. 307-XXX
© MIR Labs, www.mirlabs.net/jias/index.html

Software Watermarking Using Fixed Size Encoding and Random Dummy Method Insertion

Subariah Ibrahim[1] and Azyan Yusra Kapi[2]

[1] Department of Computer System & Communication, Faculty of Computer Science & Information System,
Universiti Teknologi Malaysia, 81310 UTM Skudai, Malaysia
subariah@utm.my

[2] Department of Computer Science, Faculty of Computer and Mathematical Sciences,
Universiti Teknologi MARA Negeri Sembilan, 72000 Beting, Kuala Pilah, Malaysia
azyanyusra@ns.uitm.edu.my

***Abstract*: The rise of software piracy has become rampant and a major concern among software developers. One of the techniques that can be used to discourage piracy is watermarking, by embedding developer's watermark into software which can later be extracted to prove ownership. During the last few years, different algorithms were produced and developed to hide the watermark inside software. This paper enhances dummy method insertion technique in embedding and recognizing the watermark in Java class files. The enhancement includes the use of fixed size encoding scheme and random dummy method insertion. The proposed fixed size encoding scheme used hash function that can produce a fixed size watermark bit sequences. Random dummy method insertion selects a dummy method from a collection of dummy methods randomly. Finally, this study analyzes the enhancement of dummy method insertion technique using two different measures, namely data-rate and resilience of the watermarking algorithm. In terms of data rate, the results show that encoded watermark for proposed encoding scheme is always fixed even though size of watermark character is increased. In terms of resilience, experimental results show no similarity between class files and thus survived from collusion attack compared to previous method.**

***Keywords*:** software watermarking, encoding scheme, hash function, software piracy, dummy method, Java class file

I. Introduction

Nowadays, software has become a part of human's daily life to ease many tasks in the software industries, e-commerce and many more. As the usages of the software are growing rapidly, the rise of software piracy has become a major concern for software developers.

Java applications that have been sold to users and distributed through the internet also suffered from piracy. While Java becomes a popular language in software industry and education, the advantages of Java which are platform-independent and its portability has create a problem towards Java's user. Platform-independent means that once Java source code is written and compiled, it can be run anywhere in any platform.

In order to make Java application executes in any platform, Java source code needs to be translated into Java class file. Based on study by [1], Java class file contains Java byte code and understandable by Java Virtual Machine (JVM). An attack to this class file can easily be made by reverse-engineering or de-compilation of the class file itself. Thus, extracting the source code from the class file is possible by many tools that can be found in the internet.

The process of reverse-engineering is become easier, thus competitors can copy other person's work easily. This could bring benefits to competitors since it is time consuming and reduce cost without having to create the algorithm themselves. Furthermore, competitors and other people could claim other developer's program as their own products.

As the technology become rampant, the process of copying other person's work become easier and it causes piracy to become crucial issues. According to the study done by the [2] in 2009 software piracy has caused USD50 billion lost in the global software industry, whereas USD368 million lost in Malaysia. In previous literature, there are many techniques to prevent software piracy. Code obfuscation, hardware based solution, checksums and watermarking are some of the example of the techniques that exist in the literature mentioned by [3].

Software watermarking is one of the techniques that can be used to prevent piracy by hiding the developer's information inside software [4]. Cappaert *et al.* [4] stated that the information can be retrieved later and used when needed to prove ownership of the original developer. For example, A is a software developer and sells his software to B. B copied the software and sells it to third parties and claimed it as his work. When A knew that B copied his product, A could use the watermark that has been embedded in the software to prove his ownership.

Considering the importance of software watermarking, many algorithms were developed in embedding and recognizing watermark [5]. Usually, watermark is translated into unreadable string before it can be embedded in the software to avoid visibility of watermark [6]. The process of converting and translating the code is known as encoding process. In this paper, we present an encoding scheme that produce a fixed size encoded watermark so that the dummy method that is used to embed the watermark is less noticeable to attackers. In addition, random dummy method is used to insert dummy method randomly from a collection of dummy methods. The purpose of random dummy method insertion is to avoid the similarity between two or more Java class file. Furthermore, with the proposed encoding scheme, long information can be used as a watermark.

The paper is organized as follows. The next section describes related works in software watermarking. Section III explains our proposed encoding scheme and random dummy method insertion then outlines each process in it. We present the results of our proposed encoding scheme and random dummy method insertion in Section IV. This paper concludes in Section V.

II. Related Works

Up until now, many algorithms such as Qu-Potkonjak (QP) algorithm, opaque predicate algorithm and many others were introduced in the literature. Despite all that, [7] summarized that unfortunately most of the algorithms on software watermarking are not well-described, not being implemented and evaluated yet. In their study, they have tested two algorithms which are dummy method insertion algorithm and Davidson–Myhrvold (DM) algorithm. Both of the algorithms have been evaluated according to the six different properties mentioned in their study.

In their evaluation, DM algorithm reveals a high credibility, satisfactory in data-rate and 50% survival rate towards resiliency. In case of dummy method algorithm, they have pointed several pros and cons in term of part protection, data-rate and also has 70% survival rate towards resiliency.

Generally in the dummy method insertion technique, all the developer's information is hidden inside the dummy method. In term of part protection, the technique or algorithm must be able to fully protect the watermark so that the whole watermark is spread throughout the entire class file. Both of DM algorithm and dummy method insertion algorithm hide the watermark in a single method. Thus, [7] concluded that DM algorithm and dummy method insertion algorithm can be considered as poor, since any alteration involved in the statement of the method will destroy the watermark.

Dummy method insertion was first introduced by Monden *et a*l. [8]. The algorithm inserts a dummy method into the software that is then used to hide the watermark. The advantages and disadvantages of dummy method insertion technique have been evaluated in [7]. One of the advantages of dummy method insertion is high data rate.

High data rate represents a good point in the algorithm as it can hide a large portion of watermark within the software. Previous encoding scheme that were used in [8] build their own translation code based on watermark character sequences. A hash table needs to be built in order to provide assignment rule for embedding process. As for dummy method algorithm, since the algorithm prepares the space for dummy method according to the size of the watermark; it has no difficulties in embedding large size watermark. Thus, no matter how large the watermark's size, the dummy method could provide spaces for the watermark. However, in contrast, the larger the watermark's size, the longer dummy method's instruction will be produced. In this situation, instructions in the dummy method become longer than expected and hence will create suspicions to the pirates. Despite the advantage in data rate, the dummy method insertion has the ability to hide various watermark sizes, but in contrast it leads to the visibility of the dummy method.

Other than data rate, resilience is one of the properties used in [7] to evaluate the algorithms. Resilience can be defined as the capability of the technique in securing the watermark, resist and resilient to the attack imposed upon them. There are four types of attacks that have been highlighted by [7] which are additive, subtractive, collusion and distortive attack. In addition to classify the attack, [9] have come out with seven different attacks towards the watermark in class file. However, [7] highlight the disadvantage of dummy method insertion technique in term of resilience which is collusion attack towards dummy method. The collusion attack consists of performing different watermark in the same class file. After decompiling and comparing both of the watermarked class file, several instructions in the class file can be seen noticeably by the attacker. The only difference in the class file can be considered as the watermark. This caused dummy method to be discovered by the attacker and still need to be improved in term of collusion attack. Thus, in this paper, we proposed an enhancement by using a fixed size encoding scheme and random dummy method insertion based on the previous method by [8].

III. Design of Fixed Size Encoding and Random Dummy Method Insertion

In order to enhance previous method, fixed size encoding and multiple dummy methods are developed and designs were discussed in following section. Section A discussed fixed size encoding scheme based on results presented in [14] and Section B described dummy method development in further detail.

A. Design of Fixed Size Encoding

A different encoding scheme from [8] was designed in order to improve performance of dummy method. As mentioned before, the previous encoding scheme produces dummy method instruction based on the size of watermark.

In software watermarking, encoding process is applied so that watermark characters can be encoded into unreadable string. It is because encoded watermark is not readable and less noticeable by the pirates instead of using watermark characters. Furthermore, the goal of encoding the watermark is to reduce the number of watermark characters embedded in the software so that any additional information added into the software is less noticeable. The flow of the process is depicted in Figure 1.

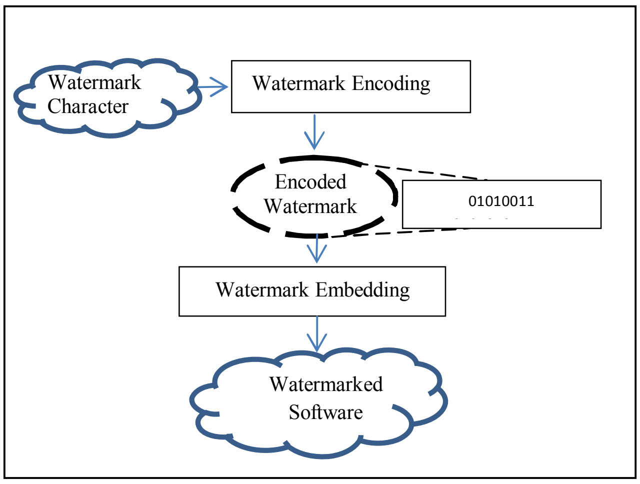


Figure 1. Flow of Software Watermarking

In Figure 1, watermark characters are input to watermark encoding process and they yield encoded watermark in a binary string format. The encoded watermark is then sent to watermark embedding process to produce watermarked software.

In case of dummy method insertion, encoding scheme shall be designed in such a way so that no matter how long the watermark's size is, it will not have impact on the dummy method's instruction. Impact on the dummy method consists of length of instructions produced on the dummy method. Thus, in designing the encoding scheme, watermark characters should be translated into fixed size of bit sequences. Conceptually, hashing function seems to be suitable for this encoding scheme. Hash function is chosen to be applied in this encoding because of its most important property, which is fixed size. It will produce a fixed size output known as message digest, no matter how big the watermark's size is. In other words it always generates the same length of message digest.

The size for bit sequence should not be too large as it degrades performance of software, neither too small to affect visibility. A 128 bit is the right answer for this purpose. After considering hash functions that fit perfectly for dummy method creation, Message Digest algorithm 5 (MD5) was selected to be used in the proposed encoding scheme since it is a well-known algorithm that could produce 128 bit message digest [10].

Thus, by using MD5 in any cases of watermark's length, it does not affect the size of the dummy method, because the size of the encoded watermark is always fixed and hence we need to design dummy method that can embed 128 bits of information for embedding watermark.

We proposed a fixed size encoding scheme that uses hash function in translating watermark characters into fixed size unreadable code which is encoded watermark. The encoded watermark is always in a fixed size according to hash function that is used in the encoding process.

B. Random Dummy Method Insertion

As mentioned in previous section, dummy method's similarity for each watermarked class file is also taken into consideration. To resolve given situation, a collection of different code representation of dummy method is generated based on [8] and will be taken randomly and inserted into class file. This will cover the similarities at least up to certain number of class file. In order to avoid the similarities between the dummy methods in watermarked class file, several designs in large collection of dummy method need to be produced. Possibility of producing similar dummy method could be avoided by providing large collection of dummy method.

Source code for dummy method is designed and compiled using Java Virtual Machine (JVM) to get a class file for the method. Class file contains byte code instruction in it. After that, embedding space in the class file is calculated. In this case, 128 bit is the exact space needed for embedding process because MD5 produced 128 bit message digest. Figure 2 shows pseudocode used in this calculation.

```
Procedure Calculate_Encoding_Space:
Input: a generic method that can be modified
BEGIN
    1.  get-instruction-list(methodGen)
    2.  if instruction-list null then
    3.  return 0
    4.  end if
    5.  for all instruction list in method loop
    6.  if instruction-list is Arithmetic then
    7.     total-encoding-space :=
         total-encoding-space + 3
    8.  else if instruction-list is Numerical then
    9.     total-encoding-space :=
         total-encoding-space + 8
    10. end loop
    11. return total-encoding-space
END
```

Figure 2. Pseudocode to Calculate Encoding Space

If the space is equal to 128, dummy method is ready to be added into collections of multiple dummy method. Otherwise, source code for the dummy method is adjusted and recalculate until 128 spaces is produced.

In order to insert dummy method to the class file, random number will be generated according to the number of multiple dummy methods. The random number represents which dummy method that needs to be inserted into class file. Figure 3 depicts process for random insertion in dummy method.

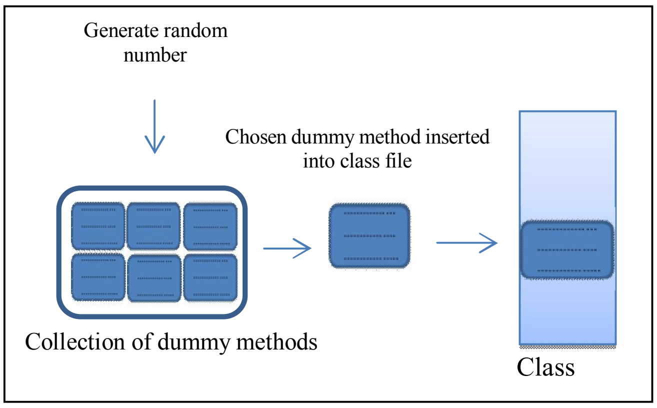

Generate random number
Chosen dummy method inserted into class file
Collection of dummy methods
Class

Figure 3. Random insertion of dummy method

From Figure 3, the dummy method was chosen randomly from the collection and inserted into the class file along with the watermark. The watermark embedding and recognition process is discussed in the following paragraph.

1) *Embedding Phase*

After encoded watermark is generated in encoding process, embedding process replaced and overwrites each bit in encoded watermark with chosen opcode in dummy method based on hash table depicted in [8]. Embedding process is done by referring to the bit assignment rules on Figure 4.

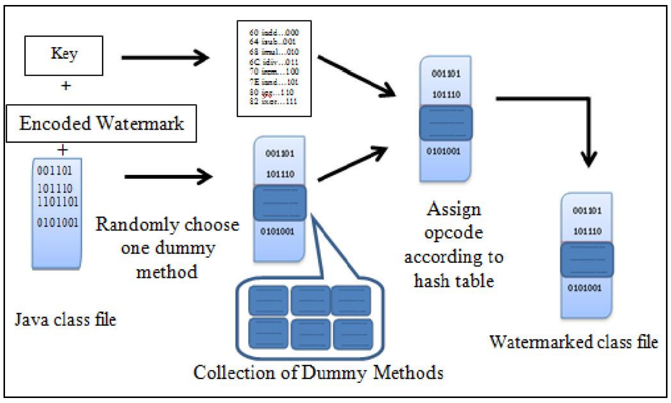


Figure 4. Design of embedding process

By referring to Figure 4, the dummy method insertion phases take Java class file as an input. Noted that to get a Java class file, Java source code must first need to go through compilation process by JVM and then translated into byte code. The byte code instructions are kept in Java class file. Process of choosing random dummy method is depicted in Figure 3.

Key generation is not a part of contributions in this study, only focused on dummy method creation and random dummy method insertion. However, key that is given by user is used to generate hash table according to bit assignment rules. The hash table describe the approach uses to embed the watermark inside the Java class file and its bit assignment rules. After bit is replaced and overwritten in dummy method, the watermarked class file is yield. Figure 5 describes step by step in embedding process.

```
Input: Class File, Watermark character, $WM_o$ and numerical key $K$

1. Encode watermark using MD5Hash and produce encoded
   watermark, $WM_e$
        Hash($WM_o$)→$WM_e$
2. Create hash table using a key, $K$
3. Choose a dummy method using Pseudo-Random Number
   Generator (PRNG)
4. Insert dummy method in class file
5. Embed $WM_e$ using hash table created in Step 2.
   5.1. Search for any numerical operands and arithmetic
        opcodes that match in hash table
   5.2. If match found for numerical operand,
        5.2.1. Overwrite numerical operand by desired 8
               bits in $WM_e$
   5.3. If match found for arithmetic opcode,
        5.3.1. Replace opcode by desired 3 bits in $WM_e$
   6. Repeat step 5.1 to 5.3 until 128 bit of $WM_e$ is
   embedded in class file
```

Figure 5. Step by step in embedding process

In software watermarking, design of watermarking system includes embedding and recognition phases. Recognition phase is described in following section.

2) *Recognition Phase*

The recognition procedure is based on [8] but with some enhancement. Watermark is retrieved from the class file based on bit assignment rules described in [8]. Retrieval is done by searching from top of the class to the end of the class file. It is assumed that bit assignment rules are known from the beginning.

The change made in this recognition phase is that in order to recognize the watermark, input of the claimed watermark must be supplied together with the class file. The claimed watermark can be identified as the original watermark that the developer claims to prove it as theirs. In the end of the recognition phase, the encoded watermark from the claimed watermark is then will be compared to the encoded watermark retrieved from the class file. If they are similar to each other, then, class file is proved as theirs. Figure 6 shows design of recognition phase for proposed method.

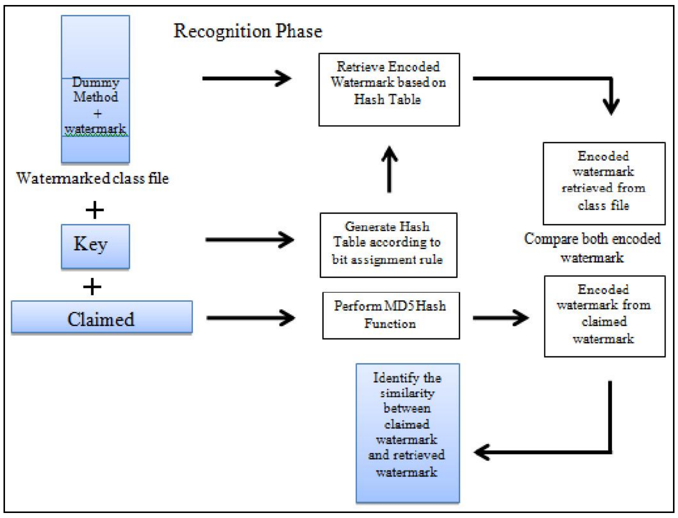

Recognition Phase
Dummy Method + watermark
Watermarked class file
+
Key
+
Claimed
Retrieve Encoded Watermark based on Hash Table
Generate Hash Table according to bit assignment rule
Perform MD5 Hash Function
Encoded watermark retrieved from class file
Compare both encoded watermark
Encoded watermark from claimed watermark
Identify the similarity between claimed watermark and retrieved watermark

Figure 6. Design of recognition phase

From Figure 6, step by step procedure in recognition phase is depicted in Figure 7.

```
Input: Watermarked Class File, Claimed Watermark character, $WM_c$ and numerical key $K$

1.  Encode claimed watermark $WM_c$ using MD5 Hash Function and this will produce encoded watermark, $WM_e$
      Hash($WM_c$) $\rightarrow WM_e$
2.  Create hash table using a key, $K$
3.  Search dummy method in class file according to method that has 128 bit embedding space and method name
4.  Retrieve original watermark $WM_o$ by using hash table created in Step 2.
    4.1  Search for any numerical operands and arithmetic opcodes that match in hash table
    4.2  If match found for numerical operand,
         4.2.1  Add desired 8 bits in $WM_o$
    4.3  If match found for arithmetic opcode,
         4.3.1  Add desired 3 bits in $WM_o$
5.  Repeat step 4.1 to 4.3 until 128 bit of $WM_o$ is retrieved from class file.
6.  Compare $WM_e$ obtained from Step 1with $WM_o$ from Step 4.
7.
```

Figure 7. Step by step in recognition process

From Figure 7, the claimed watermark is the important parameter in this phase as the original developer supposes to know their own secret information embedded in the class file. If they are original developer and owner of the watermark, the system should display the similarity of MD5 signature for both claimed and original watermark.

IV. Results and Discussions

In the previous section, design for proposed method which includes fixed size encoding scheme and random dummy method insertion has been presented. The proposed method enhances lacking properties in data rate and collusion attack discussed in Section II. Results based on data rate are presented in Section A, while Section B showed results of class file towards collusion attack.

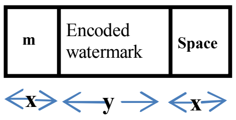
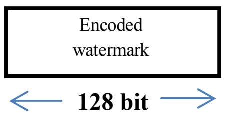
A. *Results on Fixed Sized Encoding Scheme*

In this study, the encoding scheme is designed for dummy method insertion technique, so that the size of encoded watermark is always fixed for all watermarks no matter the size of watermark to be embedded. In this case, encoded watermark can be defined as number of bits needed in embedding process. The design applied hash function and leads to a creation of fixed size for each given watermark. Thus, encoded watermark produced in proposed encoding resulted in fixed dummy method's creation compared to variation of encoded watermark. The analysis of the proposed scheme is done in three aspects: format, length of the encoded watermark and comparison between previous and proposed encoding scheme. Format of encoded watermark is discussed in the next section.

1) *Format of Encoded Watermark*

In comparison, format of encoded watermark for previous and proposed encoding is different. Our proposed encoding has eliminated prefix and suffix in previous encoding that were placed at the start and end of encoded watermark. The prefix and suffix are used to determine the starting and ending position of the watermark. Table I shows the format of encoded watermark for both encoding schemes.

Table 1. Format of Encoded Watermark for Both Encoding

Criteria	Comparisons	
	Previous Encoding	***Proposed Encoding***
Format		
Difference	Has overhead, **x**: size of prefix and suffix **y** : size of watermark	Fixed size of encoded watermark, remove overhead.

Referring to Table I, encoded watermark in proposed encoding scheme is always 128 bit in size whereas previous encoding produces variable encoded watermark. Another disadvantage is that the previous encoding scheme requires some overhead that indicate the start and end of the encoded watermark. However, proposed encoding has eliminated the overhead in the previous encoding.

2) *Length of Encoded Watermark*

Another difference between previous and proposed encoding scheme is the length of encoded watermark. Three different lengths of watermark are used as samples for the analysis which are 20, 32 and 44 characters. Table II shows the encoded watermark produced by encoding scheme for the previous and the proposed scheme.

Table 2. Comparisons of Encoded Watermark's Length for Both Encoding

Watermark Character	Encoded Watermark Using Previous Encoding	Encoded Watermark Using Proposed Method
(C) AHMAD AMRAN 2011 **LENGTH:** 20 characters	00000001001000110100 01010110010001110011 01000110100001001001 00111010101111001100 **LENGTH:** 80 bits	10111011111000010000 01100000111011111010 00001011000100000101 11110111010100110100 10000100101000011000 01100000100010000000 1110100011011 **LENGTH:** 128 bits
AHMAD AMRAN AIST SDN BHD 2009-11 **LENGTH:** 32 characters	00000001001000000011 01000000001001010000 01100100000001111000 10010100100000110110 01001010000100110100 10111100110011011110 11111111 **LENGTH:** 128 bits	01011010000000010001 10111101101010111101 11001001000010111110 11010011110001101110 11100110001100111101 01101001110010111100 1111111011001 **LENGTH:** 128 bits
(C) AHMAD AMRAN ART IN SOFTWARE SDN BHD 2011 **LENGTH:** 44 characters	00000000010001000011 00100001010011000100 00111000110010000110 01000001000100100011 00100010000101000011 01011010010001101100 01101011100101001111 00100010001000000011 01100001110100100011 10001001010011100011 10010100111010010100 **LENGTH:** 220 bits	11001000100010100010 11101101011000100010 10110001001100111111 01110110110011001100 01001111001101001010 01111110000011110111 0111111100001 **LENGTH:** 128 bits

In the previous scheme, the first example shows that 20 characters length of watermark produces 80 bits encoded watermark. By using the proposed method, the first example has resulted in producing 128 bits encoded watermark. The previous encoding has an advantage in the first example but the second example shows that both of the encoding schemes produce the same length of encoded watermark which are 128 bits. Finally, the third example in Table 2 displays 44 watermark characters produce 220 bits in the previous encoding scheme while the proposed scheme outputs only 128 bits. These three comparisons proved that the proposed method always produce fixed size of 128 bit of encoded watermark, no matter how long the watermark character is entered as an input. From the table it can be seen that that the length of the encoded watermark increases as the length of the watermark characters increases for the previous method. Thus, long watermark character has no impact on dummy method instruction technique.

3) Comparison between Previous and Proposed Encoding

In order to compare both encoding scheme in terms of data rate, a formula for calculating encoded watermark is generated for both previous and proposed encoding schemes. The previous encoding scheme uses a specific method in order to minimize encoded watermark in dummy method [8]. The limitation of space in the dummy method needs the encoded watermark to be as small as possible. In order to minimize the length of encoded watermark, the bit to be embedded in the dummy method is defined using a translation scheme [8]. The formula to calculate length of encoded watermark is based on (1).

$$B = mr + s \quad ; m = 2^p \geq q \tag{1}$$

where

B = length of encoded watermark
r = watermark's length
s = number of bits to store watermark's length
m = number of bits that accommodate distinct characters
p = number of bits to encode different characters
q = different characters in watermark

The length of encoded watermark is formulated in this study by referring to the previous encoding scheme in [8]. This formula depends on different characters in watermark, where increasing number of distinct character would increase the length of encoded watermark. By using an assumption that 8 bits is needed to store watermark's length s, length of encoded watermark in the previous encoding scheme is calculated and displayed in Table 3.

Table 3. Length of Encoded Watermark in Previous Encoding

Watermark's length, r	Different characters in watermark, q	Number of bits that accommodate distinct characters, m	Length of encoded watermark, B
16	10	4	72
18	12	5	98
20	14	5	108
22	16	5	118
24	18	5	128
26	20	5	138
28	22	5	148
30	24	5	158
32	26	5	168
34	28	6	212
36	30	6	224
38	32	6	236
40	34	6	248
42	36	6	260
44	38	6	272

In Table 3, length of encoded watermark increases as the watermark's length increases. This lead to disadvantage in dummy method insertion technique as the size of dummy method's instruction also increases. On the other hand, the formula for calculating the length of the encoded watermark for the proposed encoding scheme is as follows:

$$\mathrm{B} = c \tag{2}$$

where

B = length of encoded watermark
c = size of hash value in bit length
By using (2), the length of encoded watermark for the proposed encoding scheme is presented in Table IV. From Table 4, constant rate in producing encoded watermark in proposed method has resulted in high data rate for embedding process. High data rate is one of the important properties of watermark that represents a large capacity of information that can be hidden in software [11]. Results in Table 3 and Table 4 were presented in form of graph in Figure 3.

Table 4. Length of Encoded Watermark in Proposed Encoding

Watermark's length, *r*	Length of encoded watermark, *B*
16	128
18	128
20	128
22	128
24	128
26	128
28	128
30	128
32	128
34	128
36	128
38	128
40	128
42	128
44	128

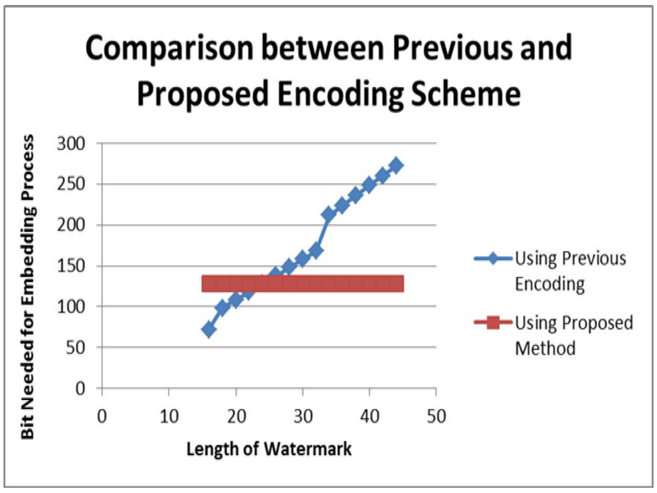


Figure 8. Comparison between Previous and Proposed Encoding

In Figure 8, it can be concluded that by using the previous scheme, encoded watermark is proportional to the number of distinct character. This is because more calculation and process is needed in order to encode the watermark character. More process such as hash table is involved in previous encoding and more time is needed in terms of performances of the system.
In order to use the previous encoding scheme, the length of watermark character needs to be stored along with the watermark. This situation has added extra length to the encoded watermark. However, by using the proposed encoding scheme, the encoded watermark is always fixed according to the size of hash value. Hash value in this proposed encoding scheme is equal to 128 bit. Even though the length of the watermark varies, after encoding process, the length of encoded watermark is fixed and always 128 bits. The size of encoded watermark leads to the number of instruction in dummy method. This situation reduces the suspicious of dummy method towards collusion attack. Collusion attack is defined as fingerprint comparison in different application [12] whereas [13] classified it as different application but contains the same watermark. In this situation, dummy method insertion technique could avoid collusion attack of different application but with the same watermark embedded.

B. Results on Random Dummy Method Insertion

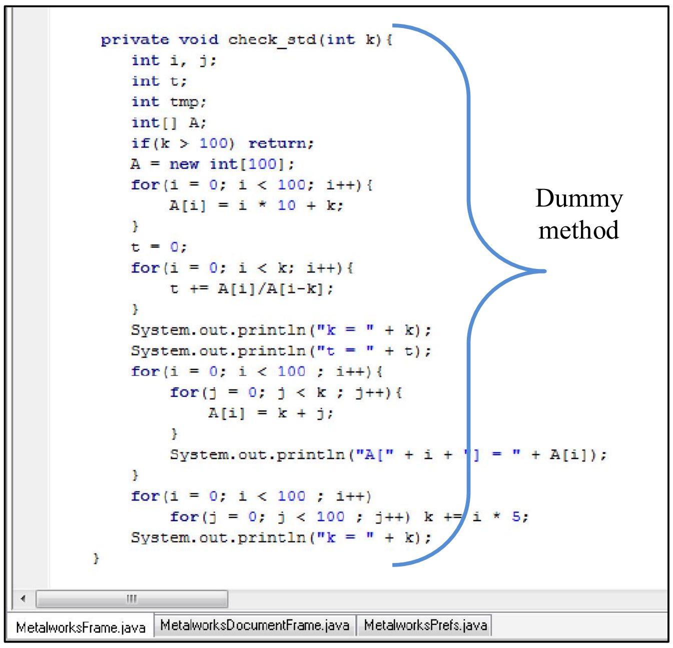
Collusion attack is done in order to compare two different class files and detect any similarities between them. This situation can be depicted in real case, if attacker tends to compare different two class files and detect existence of dummy method on it. If dummy method is detected, attacker could remove the method and watermark embedded on it will then be destroyed. After dummy method is inserted into class file, the class file is then decompiled to get the source code. This attack is done by decompiling watermarked class file using JadDecompiler. Table 5 shows the differences of instruction on dummy method for both methods in first test.

Table 5. Comparison of embedded watermark between previous method [8] and proposed method

Criteria	**Previous Method**	**Proposed Method**
Length of Embedded Watermark (characters)	20	20
Pattern in embedding	Focus only on first instruction in dummy method **Offset:** 0 – 40, 51,63, 75, 87 The rest of opcodes is copied from existing method in the class file	Distributed through several instructions in dummy method **Offset:** 1, 7, 16, 25, 27, 29, 31, 57, 59, 60, 63, 123, 140, 142, 185, 194, 202, 210, 211, 213 and 219 Does not reuse the instruction in existing method.

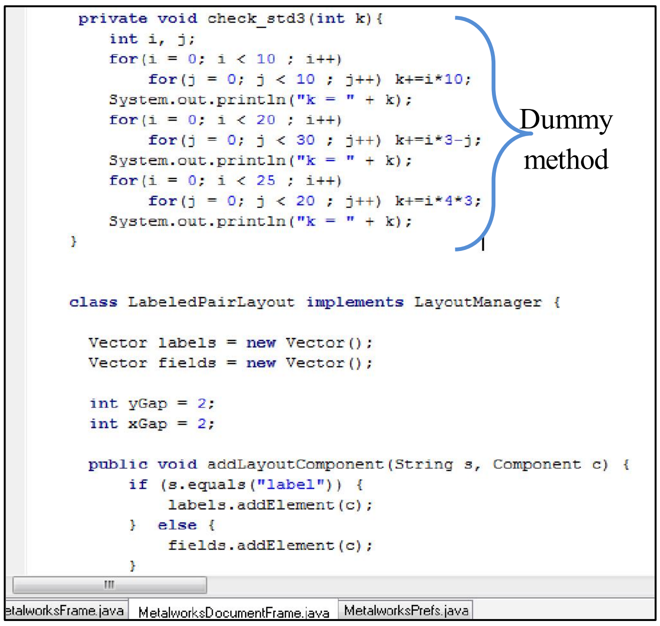
From Table 5, previous method embeds the watermark in first instruction of dummy method whereas proposed method is distributed. Besides, proposed method does not reuse instruction in existing method unlike previous method.
In random dummy method insertion technique, collusion

attack has been done to compare different test file with same watermark. As mentioned in Section I, objective of this study is to design an encoding scheme technique and to enhance a watermarking technique that is less noticeable to the attacker. Problems in previous implementation were discussed in Section II. Further testing was done to demonstrate that objectives are achieved throughout this study.
For each of test file, they contained different number of class file. Only a few of them were chosen to be watermarked. In this attack, three different watermarked files in same test files were being decompiled. Screen captured for dummy method on the three different class files were captured and depicted in Figure 9, Figure 10 and Figure 11 respectively. Figure 9 highlights dummy method on "MetalworksFrame" class file after decompiling process.



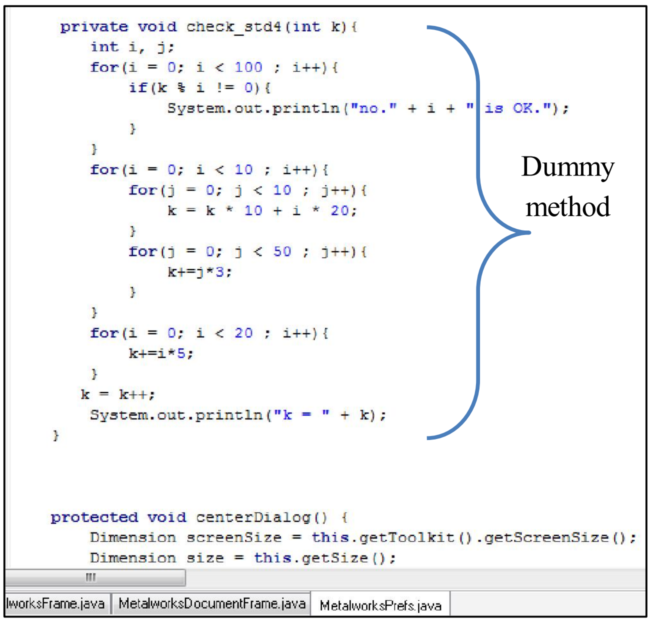
```
private void check_std(int k){
    int i, j;
    int t;
    int tmp;
    int[] A;
    if(k > 100) return;
    A = new int[100];
    for(i = 0; i < 100; i++){
        A[i] = i * 10 + k;
    }
    t = 0;
    for(i = 0; i < k; i++){
        t += A[i]/A[i-k];
    }
    System.out.println("k = " + k);
    System.out.println("t = " + t);
    for(i = 0; i < 100 ; i++){
        for(j = 0; j < k ; j++){
            A[i] = k + j;
        }
        System.out.println("A[" + i + "] = " + A[i]);
    }
    for(i = 0; i < 100 ; i++)
        for(j = 0; j < 100 ; j++) k += i * 5;
    System.out.println("k = " + k);
}
```

Figure 9. Dummy method on "MetalworksFrame" class file after decompiling process



```
private void check_std3(int k){
    int i, j;
    for(i = 0; i < 10 ; i++)
        for(j = 0; j < 10 ; j++) k+=i*10;
    System.out.println("k = " + k);
    for(i = 0; i < 20 ; i++)
        for(j = 0; j < 30 ; j++) k+=i*3-j;
    System.out.println("k = " + k);
    for(i = 0; i < 25 ; i++)
        for(j = 0; j < 20 ; j++) k+=i*4*3;
    System.out.println("k = " + k);
}


class LabeledPairLayout implements LayoutManager {

  Vector labels = new Vector();
  Vector fields = new Vector();

  int yGap = 2;
  int xGap = 2;

  public void addLayoutComponent(String s, Component c) {
      if (s.equals("label")) {
          labels.addElement(c);
      }  else {
          fields.addElement(c);
      }
```

Figure 10. Dummy method on "MetalworksDocumentFrame"class file after decompiling process



```
private void check_std4(int k){
    int i, j;
    for(i = 0; i < 100 ; i++){
        if(k % i != 0){
            System.out.println("no." + i + " is OK.");
        }
    }
    for(i = 0; i < 10 ; i++){
        for(j = 0; j < 10 ; j++){
            k = k * 10 + i * 20;
        }
        for(j = 0; j < 50 ; j++){
            k+=j*3;
        }
    }
    for(i = 0; i < 20 ; i++){
        k+=i*5;
    }
   k = k++;
    System.out.println("k = " + k);
}


protected void centerDialog() {
    Dimension screenSize = this.getToolkit().getScreenSize();
    Dimension size = this.getSize();
```

Figure 11. Dummy method on "MetalworksPrefs" class file after decompiling process

By using same watermark, two different test files in previous implementation formed same instruction in the first line of dummy method. This indicated the same watermark which is embedded in test file. Thus, in real situation, it is easier for attacker to detect the watermark if they have two different copies of applications from the same developer. However, current implementation shows that different instruction in dummy method was produced for three of class files. Even though same watermark is embedded in three different class files which are shown in Figure 9, Figure 10 and Figure 11, they depicted no similarity for dummy method instructions when compared to each other. The difference and no similarity in term of line of code could make the dummy method less noticeable to attacker compare to the same instructions that were produced in previous implementation that lead to suspicions among the attacker. Thus, proposed method results in less noticeable dummy method compared to previous implementation.

V. CONCLUSIONS AND FUTURE WORK

Software watermarking has been widely used to serve the purpose of embedding secret information of original developer and can be retrieved when needed. Software watermarking usually consists of watermark encoding, watermark embedding and watermark retrieving. In this paper, we have presented an encoding scheme that produces fixed size encoded watermark. The proposed scheme is done based on the problem that arises in dummy method insertion technique that has been discussed previously. Our proposed scheme applied a hash function and it can be useful to hide large watermark characters in watermark embedding. In conclusion, the proposed encoding scheme produces encoded watermarks of the same sizes as compared to the previous encoding scheme. Furthermore the scheme does not affect the number of lines in dummy method coding and prove to be high in data rate.

The analysis done in this study is concluded that a disadvantage in previous method is improved by creating 128 bit fix size for each watermark character. In term of data rate, graph showed that proposed method have constant rate of encoded watermark. This caused dummy method to be less noticeable during collusion attack. Thus, algorithm in embedding and extracting process can protect the copyright owner to remain on the software without being noticed by the attacker. Both situations can be compared on previous implementation. The successful of the applied method in dummy method insertion technique has contributed to the enhancement of the general technique. In future, several modifications need to be considered in order to improve and enhance the performance of the dummy method insertion technique.

In conducting an analysis, more properties need to be tested on the class file. Several attempt that suitable in attacking the class file need to be done. Additive attack for example can be done, but different test file need to be tested on proposed method. Another method shall be implemented so that it can detect the first watermark instead of second watermark embed in class file. In addition as the proposed watermark do not ensure to meet other watermark properties, it is recommended to test the algorithm in different properties. The performance of the algorithm also needs to be considered in reducing overhead issue. In conclusion, there is still a room for improvement that need to be done towards the carried study.

Acknowledgment

This work was supported by Universiti Teknologi Malaysia (UTM), Johor, Malaysia under the VOT:Q.J13000.7128.00J29.

References

[1] Lim, H., Park, H., Choi, S. and Han, T. (2009). A Method for Detecting the Theft of Java Programs through Analysis of the Control Flow Information. *Journal of Information and Software Technology* 51, September 9, 2009. 1338-1350.

[2] B. S. Alliance, "Sixth annual BSA Global and IDC global software piracy study," Business Software Alliance, Tech. Rep. 6, 2008.

[3] Naumovich, G. and Memon, N. (2003). *Preventing Piracy, Reverse Engineering, and Tampering Computer*, 36(7), 64- 71, July 2003.

[4] J. Cappaert, B. Preneel, B. Anckaert, M. Madou, and KD. Bosschere "Towards Tamper Resistant Code Encryption: Practice and Experience". Proc. of the 4th international conference on Information security practice and experience (ISPEC'08), Liqun Chen, Yi Mu, and Willy Susilo (Eds.). Springer-Verlag, Berlin, Heidelberg, 2008, pp. 86-100.

[5] C. Collberg, G. Myles and A. Huntwork, Sandmark—A Tool for Software Protection Research. IEEE Security and Privacy 1, 2003, pp. 40-49.

[6] Y. He, "Tamperproofing a Software Watermark by Encoding Constants". Department of Computer Science, University of Auckland 2002.

[7] G. Myles, C. Collberg, Z. Heidepriem, and A. Navabi, "The evaluation of two software watermarking algorithms," Journal of Software – Practice and Experience, vol. 35, pp. 923–938, 2005.

[8] A. Monden, H. Iida, K. Matsumoto, K. Inoue, and K. Torii, "A Practical Method for Watermarking Java Programs," In the 24th Computer Software and Applications Conference, pp. 191 – 197, 2000.

[9] Gupta, G. and Pieprzyk, J. (2007). Software Watermarking Resilient to Debugging Attacks (2007). *Journal of Multimedia*, 2, 10-16, Apr 2007.

[10] J. Deepakumara, H.M. Heys and R. Venkatesan, "FPGA implementation of MD5 hash algorithm". In the Electrical and Computer Engineering Conference, vol. 2, Toronto, Ont., Canada, 2001, pp. 919 – 924

[11] W. F. Zhu, "Concepts and Techniques in Software Watermarking and Obfuscation," Thesis of Doctor of Philosophy in Computer Science, Department of Computer Sciences, University of Auckland, New Zealand, 2007.

[12] J. Nagra, "Collusive Attacks against Software Watermarks". Annual International Technical Conference of IEEE TENCON Region 10 Conference. Wan Chai, Hong Kong.14-17 November, 2006.

[13] H. Zhao, "Watermark Attacks" [PowerPoint slides]. Retrieved from ENEE739M Multimedia Comm. & Info. Security, 2002.

[14] Kapi, A.Y., Ibrahim, S. , "Fixed size encoding scheme for software watermarking,", 2011 7th International Conference on Information Assurance and Security (IAS), pp.35-39, 5-8 Dec. 2011.

Author Biographies

Subariah Ibrahim received her B.Sc (Mathematic with Computer Science Minor), 1980 from University of Nevada, Reno, Nevada, USA., Masters Degree in M.Sc (Computer Science) in 1983 from Washington State University, Pullman, Washington, USA and Ph.D. (Computer Science) from University Teknologi Malaysia (2008). She was an EC-Council Network Security Administrator in 2009.

Azyan Yusra Kapi received her B.Sc Computer Science (2009) from Universiti Teknologi MARA, Shah Alam, Malaysia and Masters Degree in M. Sc. (Computer Science) in 2011 from Universiti Teknologi Malaysia.