A Survey of Software Watermarking by Code Re-Ordering

James Hamilton and Sebastian Danicic
Department of Computing
Goldsmiths, University of London
United Kingdom
james.hamilton@gold.ac.uk, s.danicic@gold.ac.uk

***Abstract*—Software watermarking involves embedding a unique identifier within a piece of software, to discourage software theft. The global revenue loss due to software piracy was estimated to be more than $50 billion in 2008. We survey the proposed software watermarking algorithms based on code re-ordering. This family of static watermarks use semantics-preserving transformations to encode a watermark in a permutation of the existing code. We describe the existing techniques and highlight the short-comings of these algorithms, namely that they are highly susceptible to semantics preserving transformations attacks.**

***Index Terms*—software watermarking; register allocation; graph colouring; program transformation;java; bytecode;**

I. Introduction

Software theft, also known as software piracy, is the act of copying a legitimate application and illegally distributing that software, either free or for profit. Legal methods to protect software producers such as copyright laws, patents and license agreements [7] do not always dissuade people from stealing software, especially in emerging markets where the price of software is high and incomes are low. Ethical arguments, such as fair compensation for producers, by software manufacturers, law enforcement agencies and industry lobbyists also do little to counter software piracy. The global revenue loss due to software piracy was estimated to be more than $50 billion in 2008 [2].

Technical measures have been introduced to protect digital media and software, due to the ease of copying computer files. Some software protection techniques, of varying degrees of success, can be used to protect intellectual property contained within Java class-files. Java bytecode is higher level than machine code and is relatively easy to decompile with only a few problems to overcome [11].

Software watermarking involves embedding a unique identifier within a piece of software. It does not prevent theft but instead discourages software thieves by providing a means to identify the owner of a piece of software and/or the origin of the stolen software [17]. The hidden watermark can be used, at a later date, to prove ownership of stolen software. It is also possible to embed a unique customer identifier in each copy of the software distributed which allows the software company to identify the individual that copied the software.

In this paper, we examine code re-ordering based software watermarking algorithms; these algorithms are static software watermarking techniques which use semantics-preserving transformations to encode a watermark in a permutation of the existing code. We report previous findings, describe some recent additions and conclude by suggesting a direction for future work.

II. Background

A. Software Watermarking

Software watermarks can be broadly divided into two categories: static and dynamic [6]. The former is embedded in the data and/or code of the program, while the latter is embedded in a data structure built at runtime.

A watermark is embedded into a computer program through the use of an *embedder*; it can then be extracted by an *extractor* or verified by a *recogniser*. The former extracts the original watermark, while the latter merely confirms the presence of a watermark [28]. A watermark recognition or extraction algorithm may also be classified as *blind*, where the original program and watermark is unavailable, or *informed*, where the original program and/or watermark is available [27].

Watermarks should be resilient to semantics preserving transformations and ideally it should be possible to recognise a watermark from a partial program. Semantics preserving transformations, by definition, result in programs which are syntactically different from the original, but whose behaviour is the same. The attacker can attempt, by performing such transformations, to produce a semantically equivalent program with the watermark removed. Redundancy and recognition with a probability threshold may help with these problems [16].

The runtime cost of a program with an embedded watermark should not differ significantly from the original program but some transformations applied by the watermark could have an effect on size and execution time [19].

B. Watermarking by Permutation

It is well known that information can be hiding in the order of lists [26] and this can be exploited for watermarking purposes. Given a list of size n we can store $log_2 n!$ bits as there are $n!$ permutations of the items in n.

TABLE I: Factorial Numbers

base	8	7	6	5	4	3	2	1
place value	7!	6!	5!	4!	3!	2!	1!	0!
in decimal	5040	720	120	24	6	2	1	1

The basic idea is to convert the watermark into an integer W and re-order the list l as the W^{th} permutation of l. For example, to embed the watermark number 22 into the list $\{1, 2, 3, 4\}$ we have to find the 11^{th} permutation.

The factorial number system is useful for finding permutations of lexicographically ordered sets. Factorials are used as the place values in the factorial number system, as shown in table I.

For example, to convert the decimal number 22 into the factorial number system:

$$22_{10} = 3 \times 6 + 2 \times 2 + 0 \times 1 + 0 \times 0 = 3200_!$$

To find the 22^{nd} permutation of $\{1, 2, 3, 4\}$, each of the factorial number digits acts as an index (starting at position 0) which shows the number to remove next to build up the permutation set.

```
factoradic:  3          2       0       0
             |          |       |       |
      (1,2,3,4) -> (1,2,3) -> (1,2) -> (2)
             |          |       |       |
permutation:(4,         3,      1,      2)
```

The 22^{nd} permutation is therefore $\{4, 3, 1, 2\}$.

III. Basic Block Re-Ordering

Davidson and Myhrvold [8] proposed one of the first software watermarking algorithms which encodes the watermark by basic block re-ordering. The embedding algorithm was described in a patent issued to Microsoft but the extraction algorithm was not discussed. Collberg *et al.* [18] proposed a method of watermark extraction and implemented the DM algorithm in Sandmark [4].

Definition 1. *Basic Block [1]: A basic block is a sequence of consecutive statements in which the flow of control enters at the beginning and leaves at the end without halt or possibility of branching except at the end.*

Collberg *et al.*'s extraction algorithm is an *informed* extraction algorithm; that is, it requires the original P and the watermarked program P^w to extract the watermark w. The embedding algorithm re-orders only unique basic blocks in all methods as there is no way of knowing which method(s) the watermark is stored in. This would result in the extraction of many watermarks; Collberg *et al.* overcome this in their implementation by prefixing and suffixing magic numbers to the watermark to guarantee recognition.

Definition 2. *Unique Basic Block [18]: A basic block is unique if and only if no other block in the graph contains the same instructions.*

Listing 1: Bubble Sort in Java

```
public static void bubbleSort(int[] arr) {
    boolean swapped = true;
    int j = 0;
    int tmp;
    do {
      swapped = false;
      j++;
      for (int i = 0; i < arr.length - j; i++) {
          if (arr[i] > arr[i + 1]) {
            tmp = arr[i];
            arr[i] = arr[i + 1];
            arr[i + 1] = tmp;
            swapped = true;
          }
      }
    } while(swapped);
}
```

Non-unique basic blocks can be made unique by inserting bogus code (such as *no op* instructions) until all the basic blocks are unique [5].

The first step in the DM algorithm is to convert the watermark into a number w; then the w^{th} permutation [14] of a set of basic blocks B is generated. The permutated basic blocks B' are re-linked to retain the original program semantics and B is replaced by B' to produce the watermarked program P'.

To extract a watermark the first step is to compare the ordering of the original basic blocks against the new ordering, to obtain the permutation number; this number is then converted back into the watermark number.

A program method containing n unique basic blocks can embed $\lfloor log_2 n! \rfloor$ watermark bits. The method should not contain exception handling code as this can impose an ordering of basic blocks which is difficult or impossible to alter [18].

Figure 1(a) shows a linearised control flow graph[1] [1] for the Java bubble sort program in listing 1. The program method contains 7 unique basic blocks (excluding *begin* and *end*) which means that this method can store 12 bits.

Figure 1(b) shows the control flow graph after embedding the watermark 1011011010_2 (730_{10}). The linearised set of basic blocks for the control flow graph is $B = \{B_0, B_1, B_2, B_3, B_4, B_5, B_6\}$ of which the 730^{th} permutation is $B' = \{B_1, B_0, B_2, B_4, B_6, B_3, B_5\}$. The blocks in B' are relinked to retain the original control flow order by inserting additional *goto* statements where neccesary, for example at the end of B_0.

Hattanda *et al.* [13] evaluated the DM watermarking algorithm by watermarking several C programs and analysing metrics such as program size and program performance. In their implementation they found that the size increase of a watermarked program was between 9% and 24% while the performance was 86% to 102% of the original program. The 2% performance increase was due to the re-ordered code containing no redundant jump instructions, and that it showed

[1]The code shown is Jimple [20] - a 3-address-code intermediate representation for Java bytecode, used by the Soot optimiser [24]

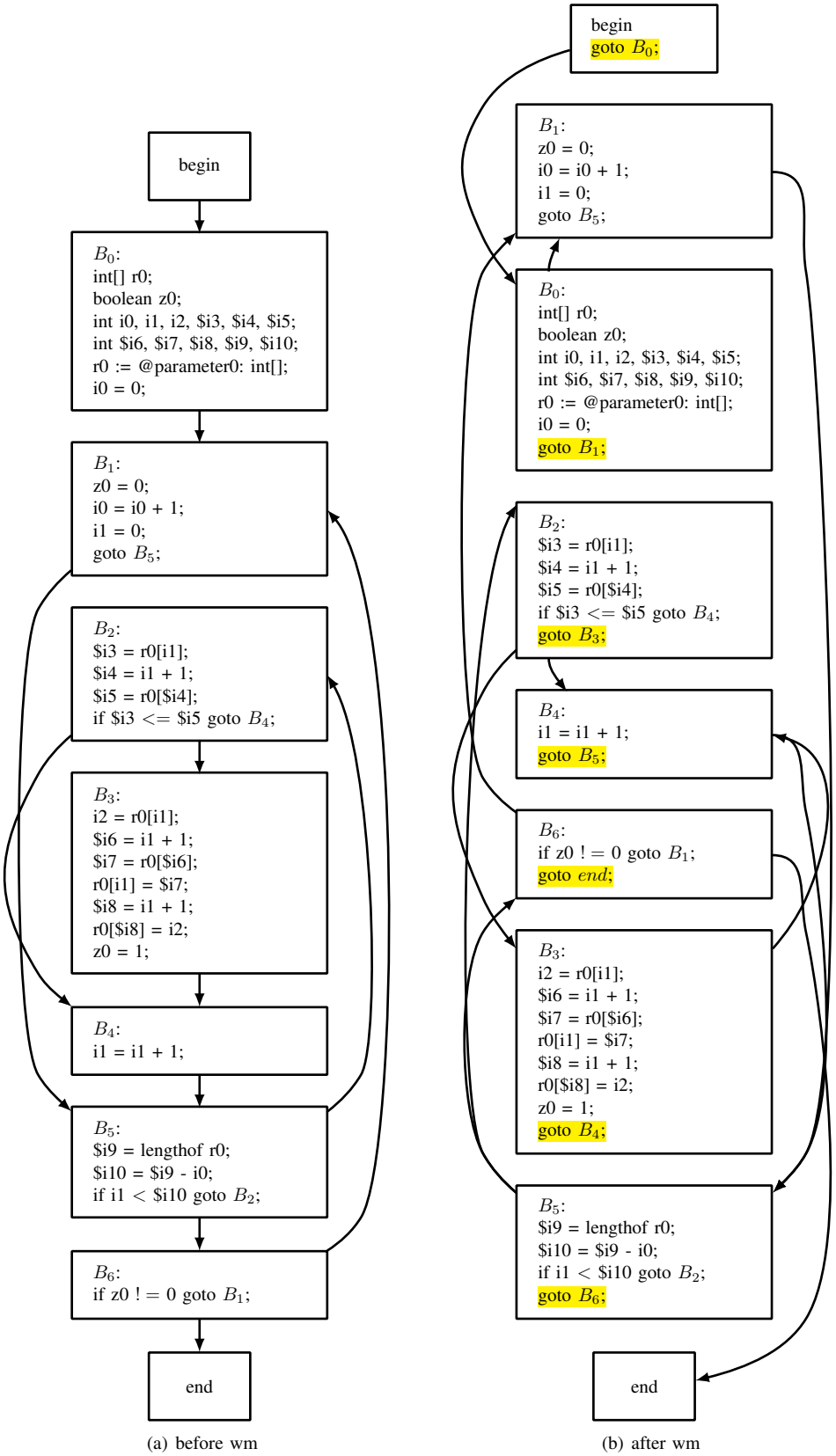

Fig. 1: Linearised Control Flow Graphs for the Java Bubble Sort program, listing 1

more locality than the original - greater locality increases performance and increasing locality is a common optimisation performed by compilers [9].

Hattanda *et al.* reported a data-rate of approximately 0.2% of program size (in bytes) based on their implementation that used a partial permutation scheme, which only used 6 basic blocks. Collberg *et al.*'s implementation was not constrained in this way and the data rate is dependent on the number of basic blocks in program methods.

The DM algorithm is highly unstealthy due to the fact that a normal compiler would not linearise the control flow graph as in DM watermarked programs [5]. A simple way to discover a DM watermark is to examine the ratio of *goto* statements to the total number of instructions - programs with the DM watermark show a high ratio compared to an unwatermarked program [18] (you can clearly see the difference in the number of *goto* statements in figure 1).

The biggest flaw with the DM watermark is that it is highly *fragile*; that is, it is not resilient to semantics-preserving transformations. For example, any transformation which re-ordered the basic blocks would eliminate the watermark [12].

Anckaert *et al.* [23] implemented and evaluated a version of the DM watermarking algorithm for machine code where groups of chains of basic blocks are re-ordered.

Definition 3. *Basic Block Chain [3]: A set of basic blocks that must be placed consecutively.*

They concluded that their watermarking algorithm is stealthier as it has a minimal affect on code locality.

IV. Equation Re-Ordering

Shirali-Shahrez *et al.* [22] proposed a software watermark scheme based the re-ordering of operations in mathematical equations. The idea involves re-ordering symmetric mathematical operations, such as addition, to preserve program semantics.

For example, equation 1 and equation 2 are equivalent and we could consider a 0 bit encoding for the first ordering, and a 1 bit encoding for the second ordering.

$$x = y + z \tag{1}$$

$$x = z + y \tag{2}$$

Not all operations are symmetric and the watermarking algorithm only re-orders *safe swappable* binary operations to ensure the watermarked equation is equivalent.

Definition 4. *Safe swappable operation [22]: An operation is safe swappable if it is symmetric and at least one of it's operands is constant.*

In order to produce a *blind* watermarking extraction algorithm an ordering is defined on the operands of a safe swappable operation:

1) If both operands are constant they are ordered according to their string representation.

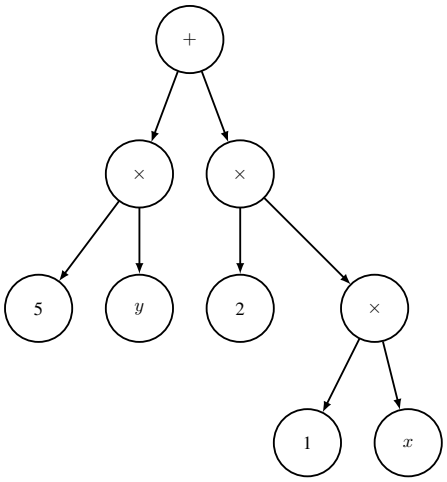

Fig. 2: Equation tree for equation 4

2) If one operand is constant and the other is not then the constant operand would come first.

The sorted order of operands is retained to encode a 0 bit whereas the operands are reversed to encode a 1 bit. The extraction algorithm can then check to see if operands are in sorted order or not, to obtain 0 or 1.

Equation 3 contains 3 safe swappable operations: $5 \times y$, $x + 1$ and $2 \times (x + 1)$.

$$5 \times y + 2 \times (x + 1) \tag{3}$$

Of these 3 operations $x + 1$ is unordered according to the ordering definition; therefore the equation must be re-ordered as (4) before watermarking.

$$5 \times y + 2 \times (1 + x) \tag{4}$$

In order to encode a watermark we perform a pre-order traversal of the equation tree (see figure 2), swapping operations where necessary to encode bits. For example to encode the watermark 011_2 we leave the order of $5 \times y$ and change the order of $2 \times (1 + x)$ and $1 + x$, giving us equation 5.

$$5 \times y + (x + 1) \times 2 \tag{5}$$

The data-rate of this watermarking technique is related to the number of safe-swappable operations within a program. It is likely that there will be many, but the watermark will probably need to be split into pieces as most equations will only encode a small number of bits individually.

Zonglu *et al.* [21] proposed a very similar technique using a re-ordering based on operand coefficients. Neither of these techniques cannot be applied to source-code as the compiler itself may re-order the operands and even when applied to bytecode or machine code this technique is highly susceptible to semantics-preserving transformations. Any re-ordering of the operands after watermarking will remove the watermark.

V. Constant Pool Re-Ordering

Gong *et al.* [10] proposed a watermarking scheme, CPW, for Java based on the ordering of a class file's constant pool.

A Java compiler compiles Java source code into intermediate Java Byte Code in files ending with the extension *.class*.

Java class files are divided into 10 areas:

Magic Number
: 0xCAFEBABE

Version Numbers
: The minor and major versions of the class file

Constant Pool
: Pool of constants for the class

Access Flags
: e.g. abstract, static, etc

This Class
: The name of the current class

Super Class
: The name of the super class

Interfaces
: Any interfaces in the class

Fields
: Any fields in the class

Methods
: Any methods in the class

Attributes
: Any attributes of the class

The constant pool is an array of variable length elements containing every constant used in the Java class [25]. Constants are referenced by an index throughout the bytecode. Some entries in the constant pool are *direct* references to a constant while other entries are references to other members in the constant pool - these are *indirect* constants. Each constant in the pool is preceded by a tag denoting it's type, for example *class*, *integer*, *Utf8* [15].

The CPW scheme involves re-ordering the *direct constants* corresponding to the W^{th} permutation of the direct constants where W is an integer watermark.

Figure 3 shows a conceptual diagram of listing 2; the constant pool is shown before and after watermarking. The set of direct constants D before watermarking is $D = \{7, 8, 9, 10, 11, 12, 13, 14, 18, 21, 22, 23, 24, 25, 26, 27, 28\}$.

If we want to embed the watermark 1011011010_2 (730_{10}) we first obtain the 730^{th} permutation of D; this is $D' = \{7, 8, 9, 10, 11, 12, 13, 14, 18, 21, 23, 22, 24, 26, 28, 25, 27\}$. Figure 3 shows the constant pool after watermarking, on the right, where the direct constants have been re-numbered according to the permutation.

To extract the watermark we calculate the original ordering of the direct constants (in the same way as the original Java compiler did) and compare with the watermarked constant pool to find the permutation number W.

Gong *et al.* evaluated their algorithm by embedding watermarks into 4000 random class files downloading from the Internet and concluded that it has a good robustness but small capacity. However, the CPW algorithm is far from robust - any further re-ordering of the constant pool would destroy the watermark.

Listing 2: Hello World in Java

```
public class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello World");
  }
}
```

VI. Conclusion

We have presented a survey of software watermarking schemes based on code re-ordering. This family of watermarks are highly susceptible to semantics-preserving transformation attacks [12], and can be unstealthy. The DM watermark is highly unstealthy due to addition of large numbers to *goto* statements inserted to preserve the original control-flow after re-ordering.

Any further re-ordering of a program watermarked with any of the presented algorithms will likely destroy the watermark. In fact, even re-watermarking the program will likely destroy the watermark.

Academic research continues in this area, with the latest paper published last year [21]. We believe that further research should instead focus on dynamic software watermarking techniques which, in theory, should be resilient to semantics-preserving transformations; thus providing a robust technique for the protection of software.

References

[1] Alfred V Aho, Monica S Lam, Ravi Sethi, and Jeffrey D Ullman. *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Addison Wesley, August 2006. Published: Hardcover.

[2] Business Software Alliance. Sixth annual BSA and IDC global software piracy study. Technical Report 6, Business Software Alliance, 2008.

[3] Bertrand Anckaert, Bjorn De Sutter, and Koen De Bosschere. Steganography for executables. *Information Security and Cryptology - ICISC 2004*, pages 425–439, 2005.

[4] Christian Collberg. Sandmark, August 2004. Available from: http://www.cs.arizona.edu/sandmark/.

[5] Christian Collberg and Jasvir Nagra. *Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection*. Addison-Wesley Professional, 2009.

[6] Christian Collberg and Clark Thomborson. Software watermarking: Models and dynamic embeddings. In *Principles of Programming Languages 1999, POPL'99*, January 1999. Available from: http://www.cs.auckland.ac.nz/collberg/Research/Publications/CollbergThomborsonLow99a/index.html.

[7] Gareth Cronin. A taxonomy of methods for software piracy prevention. Technical report, Department of Computer Science, University of Auckland, New Zealand, 2002.

[8] Robert Davidson and Nathan Myhrvold. Method and system for generating and auditing a signature for a computer program, June 1996. Microsoft Corporation, US Patent 5559884.

[9] Robert I Davidson, Nathan Myhrvold, Keith Randel Vogel, Gideon Andreas Yuval, Richard Shupak, and Norman Eugene Apperson. Method and system for improving the locality of memory references during execution of a computer program, September 2001.

[10] Daofu Gong, Fenlin Liu, Bin Lu, Ping Wang, and Lan Ding. Hiding information in java class file. In *Proceedings of the 2008 International Symposium on Computer Science and Computational Technology - Volume 02*, pages 160–164. IEEE Computer Society, 2008. doi: 10.1109/ISCSCT.2008.231.

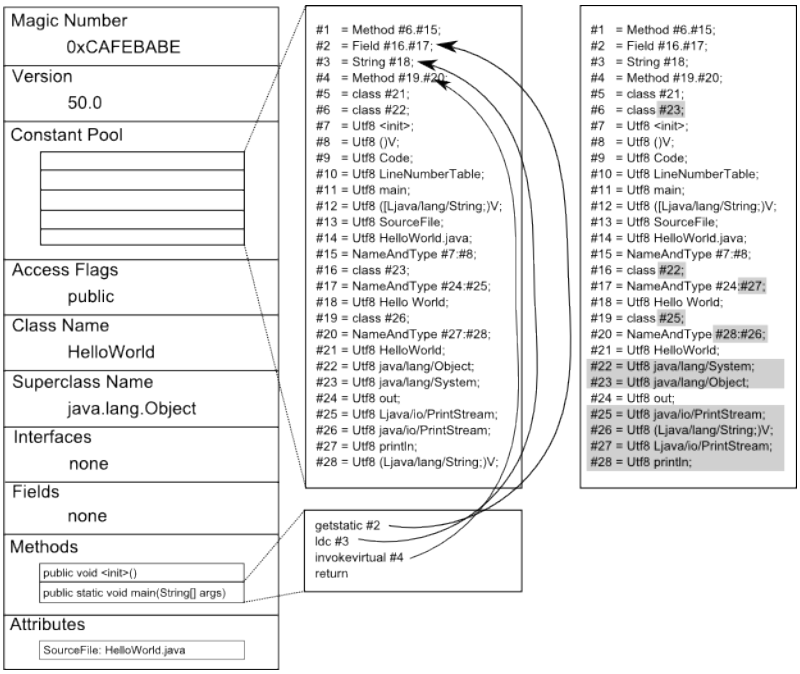

Fig. 3: Conceptual Java class file diagram, showing constant pool before and after watermarking.

[11] James Hamilton and Sebastian Danicic. An evaluation of current java bytecode decompilers. In *Ninth IEEE International Workshop on Source Code Analysis and Manipulation*, volume 0, pages 129–136, Edmonton, Alberta, Canada, 2009. IEEE Computer Society. doi:http://doi.ieeecomputersociety.org/10.1109/SCAM.2009.24.

[12] James Hamilton and Sebastian Danicic. An evaluation of static java bytecode watermarking. In *Proceedings of the International Conference on Computer Science and Applications (ICCSA'10)*, The World Congress on Engineering and Computer Science (WCECS'10), San Francisco, October 2010. To appear.

[13] Kazuhiro HATTANDA and Shuichi ICHIKAWA. The evaluation of davidsons digital signature scheme. *IEICE TRANS. FUNDAMENTALS*, E87A(1), January 2004.

[14] Donald E. Knuth. *The Art of Computer Programming, Volume 1 (3rd ed.): Fundamental Algorithms*, volume 1. Addison Wesley Longman Publishing Co., Inc., 3 edition, 1997.

[15] Tim Lindholm and Frank Yellin. *The Java(TM) Virtual Machine Specification (2nd Edition)*. Prentice Hall PTR, April 1999. Published: Paperback.

[16] Anshuman Mishra, Rajeev Kumar, and P. P. Chakrabarti. A method-based Whole-Program watermarking scheme for java class files. 2008. Available from: http://citeseerx.ist.psu.edu/viewdoc/summary10.1.1.116.2810.

[17] Ginger Myles. Using software watermarking to discourage piracy. *Crossroads - The ACM Student Magazine*, 2004. Available from: http://www.acm.org/crossroads/xrds10-3/watermarking.html.

[18] Ginger Myles, Christian Collberg, Zachary Heidepriem, and Armand Navabi. The evaluation of two software watermarking algorithms. *Softw. Pract. Exper.*, 35(10):923938, 2005. Available from: http://dx.doi.org/10.1002/spe.v35:10, doi:10.1002/spe.v35:10.

[19] Jasvir Nagra, Clark Thomborson, and Christian Collberg. A functional taxonomy for software watermarking. In Michael J. Oudshoorn, editor, *Aust. Comput. Sci. Commun.*, pages 177–186, Melbourne, Australia, 2002. ACS.

[20] Raja V Rai and Laurie J Hendren. Jimple: Simplifying java bytecode for analyses and transformations. Technical report, Sable Research Group, McGill University, Montreal, Quebec, Canada, 1998.

[21] Zonglu Sha, Hua Jiang, and Aicheng Xuan. Software watermarking algorithm by coefficients of equation. *Genetic and Evolutionary Computing, International Conference on*, 0:410–413, 2009. doi:http://doi.ieeecomputersociety.org/10.1109/WGEC.2009.18.

[22] M. Shirali-Shahreza and S. Shirali-Shahreza. Software watermarking by equation reordering. In *Information and Communication Technologies: From Theory to Applications, 2008. ICTTA 2008. 3rd International Conference on*, pages 1–4, 2008. Available from: http://dx.doi.org/10.1109/ICTTA.2008.4530357, doi:10.1109/ICTTA.2008.4530357.

[23] Bjorn De Sutter, Koen De Bosschere, and Bertrand Anckaert. Covert communication through executables. In *Program Acceleration through Application and Architecture Driven Code Transformations: Symposium Proceedings*, pages 83–85, 2004.

[24] Raja Valle-Rai, Phong Co, Etienne Gagnon, Laurie Hendren, Patrick Lam, and Vijay Sundaresan. Soot - a java bytecode optimization framework. In *CASCON '99: Proceedings of the 1999 conference of the Centre for Advanced Studies on Collaborative research*, page 13. IBM Press, 1999.

[25] Bill Venners. *Inside the Java Virtual Machine*. McGraw-Hill, Inc., New York, NY, USA, 1996.

[26] Peter Wayner. *Disappearing Cryptography: Information Hiding: Steganography and Watermarking*. Software Engineering and Programming. Morgan Kaufmann Publishers Inc., 3 edition, 2008.

[27] William Zhu. Informed recognition in software watermarking. In *Proceedings of the 2007 Pacific Asia conference on Intelligence and security informatics*, pages 257–261, Chengdu, China, 2007. Springer-Verlag.

[28] William Feng Zhu. *Concepts and Techniques in Software Watermarking and Obfuscation*. PhD thesis, The University of Auckland, 2007.