A Survey of Software Watermarking by Register Allocation (for Java Bytecode)

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 register allocation based algorithms for software watermarking. This family of static watermarks are constraint-based and embed the watermark in a solution to a graph colouring problem; the graph colouring is then used to allocate registers. Register allocation is the process of allocating program variables to a finite number of CPU registers and a compiler commonly uses graph colouring to obtain the best allocation. We describe the existing techniques and highlight the short-comings of these algorithms, namely that they are highly susceptible to semantics preserving transformations.**

***Keywords*-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 [1] 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 [3].

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 [4]. 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 register allocation based software watermarking algorithms; these algorithms are constraint-based static software watermarking techniques. Figure 14 shows the evolution of this family of algorithms on which we report previous findings, describe some recent additions (including a correction to a published algorithm) 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 [5]. 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 [6]. A watermark recogntion or extraction algorithm may also been classified as *blind*, where the original program and watermark is unavailable, or *informed*, where the original program and/or watermark is available [7].

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 [8].

The runtime cost of a program with an embedded watermark should not differ signifcantly from the original program but some transformations applied by the watermark could have an effect on size and execution time [9]. For example, Hattanda *et al.* [10] found that the size of a program, watermarked with Davidson/Myhrvold [11] algorithm,

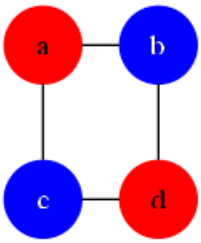

Figure 1: A graph with 4 vertices and 2 colours

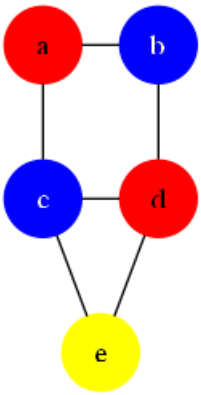

Figure 2: Example graph with 5 vertices and 3 colours

increased by up to 24% and the performance decreased by up to 24%.

Register allocation based software watermarking algorithms are static software watermarking techniques which do not embed any extra code but add redundancy to the program by changing the use of registers.

B. Graph Colouring

In graph theory, the simplest form of graph colouring is a way of colouring the vertices of a graph such that no two adjacent vertices share the same colour. The graph colouring problem is NP-complete [12].

Consider the simple graph in figure 1; the graph contains four vertices and four edges. Vertex a is adjacent to vertices b and c therefore if a is red then b and c cannot be. Vertex d, on the other hand, is not adjacent to a and can therefore also be coloured red. Vertices b and c are therefore coloured blue. The smallest number of colours needed to colour this graph is 2. This is known as the *chromatic number*.

If we add another vertex, e, adjacent to c and d we must colour this with a third colour; e cannot be blue as it is adjacent to c and it cannot be red as it is adjacent to d (figure 2). The chromatic number of graph 2 is therefore 3.

Graph colouring has many real-world applications including register allocation in compilers [13].

Listing 1: 'Example Pseudocode'

```
v1 := 1 + 1;
v2 := 2 + 5;
v3 := 3 + v1;
v4 := v1 + v2;
v5 := v2 + v3;
```

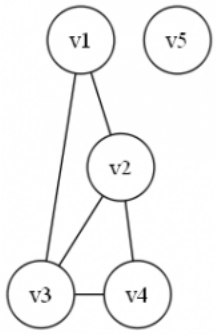

Figure 3: Example interference graph for listing 1

C. Register Allocation

Register allocation is the process of assigning program variables to a finite number CPU registers [14]. Programmers can use any number of variables in their programs but there are a limited number of CPU registers to actually store the values of those variables.

An interference graph is used to model the relationship between the variables in a program method. Each vertex in the graph represents a variable and an edge between two variables indicates that their live ranges overlap. Simple put, a variable is live if it has been computed and will be used before being recomputed. We colour the graph in order to minimise the number of registers required and ensure that two live variables do not share a register.

Definition 1. *Variable liveness [13]: A program variable is considered live at point L in a program P if there is a control flow path from the entry point of P to a definition of X and then through L to a use of X at point U, which has the property that there is no redefinition of X on the path between L and the use of X at U.*

Consider the example pseudocode in listing 1 and it's interference graph in 3. From the pseudocode we can see that variable v5 is not live at the same time as any other variable and is therefore unconnected in the graph. Variable v1, on the other hand, is live at the same time v2 and v3 are live; therefore they are connected in the interference graph.

The graph in figure 3 can be coloured, as shown in figure 4, using 3 colours. This means that the minimum number of registers needed to store the variables in the sample program is 3. Variables v1, v4 and v5 can use the same register because they are not live at the same time.

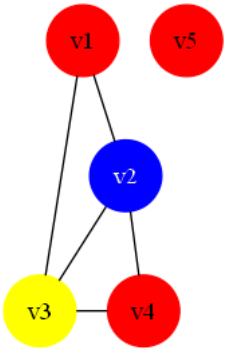

Figure 4: Example coloured interference graph for listing 1

Listing 2: 'Example Pseudo-Assembly code'

```
add r1, 1, 1
add r2, 2, 5
add r3, 3, r1
add r1, r1, r2
add r1, r2, r3
```

The sample program could be converted into pseudo-assembly code, shown below, where *add X, Y, Z* adds together Y and Z and stores the result in X. Y or Z could be a literal value or the value in a register.

We can see that r1, r2 and r3 correspond to the three colours in the interference graph.

D. The Java Virtual Machine

The Java Virtual Machine is essentially a simple stack based machine which can be separated into five parts: the heap, program counter registers, method area, Java stacks and native method stacks [15] . The Java Virtual Machine Specification [16] defines the required behaviour of a Java virtual machine but does not specify any implementation details. Therefore the implementation of the Java Virtual Machine Specification can be designed different ways for different platforms as long as it adheres to the specification. A Java Virtual Machine executes Java bytecode in class files conforming to the class file specification which is part of the Java Virtual Machine Specification [16] and updated for Java 1.6 in JSR202 [17].

An advantage of the virtual machine architecture is portability - any machine that implements the Java Virtual Machine Specification [16] is able to execute Java bytecode hence the slogan "Write once, run anywhere" [18]. Java bytecode is not strictly linked to the Java language and there are many compilers, and other tools, available which produce Java bytecode [19], [20] such as the Jikes compiler, Eclipse Java Development Tools or Jasmin bytecode assembler. Another advantage of the Java Virtual Machine is the runtime type-safefy of programs. These two main advantages are properties of the Java Virtual Machine not the Java language

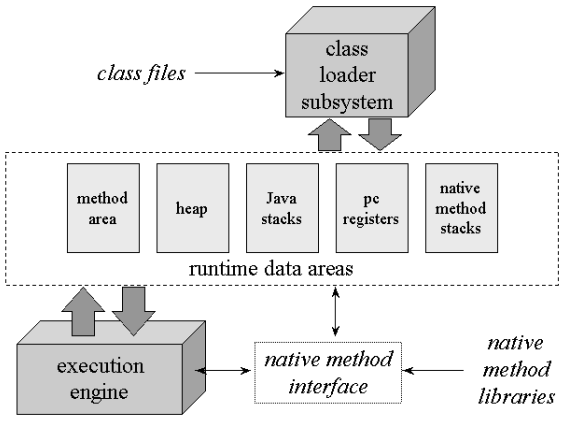

Figure 5: Java Virtual Machine internal architecture [15]

[20] which, combined, provides an attractive platform for other languages.

Java stacks are composed of stack frames, each of which contains the state of one Java method invocation. The Java virtual machine contains no registers but each method has access to a local variable table. The compiler has to allocate each Java method's variables to local variable slots. Watermarking by register allocation can be implemented by changing the uses of the local variable slots. A watermark could be included in each method in a Java program, or the watermark could be split and spread throughout the program.

III. The QP Algorithm

The QP algorithm [21] is a constraint-based watermarking algorithm based on the concept of graph colouring. In the QP algorithm edges are added to the graph based on the value of the watermark. When edges are added to an interference graph the vertices that become connected must be re-coloured - and they cannot be assigned the same registers. In other words, we add a new constraint to the problem (the extra edges) and when we compute the graph colouring we have a solution which solves the graph colour problem and one which also includes the watermark.

The QP algorithm was proprosed to embed watermarks in any graph colouring solution and can be applied to graph colouring for register allocation to embed watermarks in software.

The first step in the QP algorithm is to convert the message into binary, for example convert a string into binary by using ASCII codes. The next step is to add additional edges to the interference graph, in such a way that the extra edges encode the watermark. The QP algorithm requires that the vertices of the graph are indexed and relies on the ordering of such indices for embedding and extraction. The originally published QP algorithm contained a minor

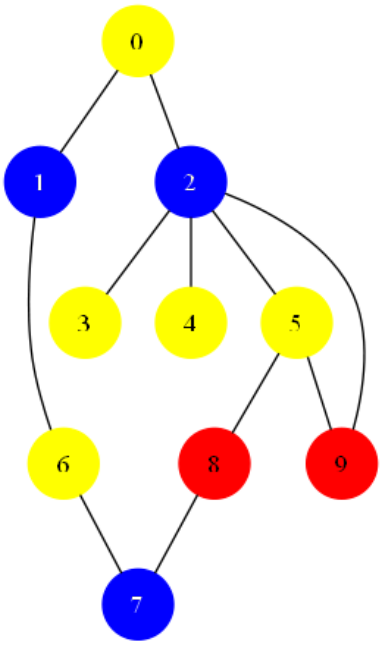

Figure 6: Example interference graph

problem which assumed that every vertex in an arbitrary graph could contain one bit of information; Zhu *et al.* proposed a clarified version of the algorithm [22] shown in algorithm 1.

Algorithm 1 Clarified QP embedding algorithm

Input: a graph $G(V, E)$, a message $M = m_0 m_1 \ldots$
Output: a graph $G'(V', E')$ with embedded message M

```
copy G(V, E) to G'(V', E')
j = 0
for each bit in message as m_i do
  if v_i has two candidate vertices v_i1, v_i2 then
    j++
    if m_j = 0 then
      add edge (v_i, v_i1) to E'
    else
      add edge (v_i, v_i2) to E'
    end if
  end if
end for
return G'(V', E')
```

Definition 2. *QP candidate vertices [21]: The nearest two vertices v_{i1} and v_{i2} which are not connected to v_i; $i_2 > i_1 > i \ (mod\ n)$ and $(v_i, v_{i1}), (v_i, v_{i2}) \notin E$ and $(v_i, v_j) \in E$ for all $i < j < i_1, i_1 < j < i_2$*

Figure 6 shows an example intereference graph for a program (it is not important which program, in these examples). In order to embed the watermark 1011011010_2 in the interference graph we use algorithm 1. Figure 7 shows the interference graph with dotted watermark edges; for clarity, these lines are labeled with the watermark value.

For each message bit we take the vertex with the corresponding index and look at the next two vertices which are not connected; if the message bit is 0 we add an edge between the current vertex and the next unconnected vertex, otherwise we add an edge to the second unconnected vertex.

For example, the first message bit is 1. Vertex v_0 is connected to vertices v_1 and v_2 therefore the next two unconnected vertices are v_3 and v_4. The message bit to embed is 1 so we add an edge between v_0 and v_4. The next two unconnected vertices after v_9 are v_0 and v_1; we add an edge between v_9 and v_0 because the watermark bit is 0.

The chromatic number of the original interference graph (figure 6) is 3, while the chromatic number of the watermarked graph is 4.

In order to extract the secret message we consider all pairs of coloured vertices which do not have edges between them. For each pair v_i, v_j we count how many vertices there are with indices between i and j which are not connected to v_i.

Definition 3. *Value of the watermark bit in the QP extraction algorithm [23]:*

1) *If $n(i, j)$ = 0, the watermark bit is 0*
2) *if $n(i, j)$ = 1, the watermark bit is 1*
3) *otherwise, if $n(j, i)$ = 0, the watermark bit is 0; if $n(j, i)$ = 1, the watermark bit is 1; else the watermark bit is undefined.*

where $n(i, j)$ is the number of vertices between v_i and v_j which are not connected to v_i.

For example, consider our watermarked graph in figure 7, in order to extract the watermark we would have to consider only the colours; the dotted edge information would be unavailable to us in practise. Between the pair (v_0, v_4) there is one vertex, v_3, which is unconnected to v_0; thus the message bit is 1.

Qu and Potkonjak include two further embedding algorithms: selecting a maximal independent set (MIS) and adding vertices & edges. They also perform an experimental evaluation to compare the difficulty of colouring the original graphs vs. watermarked graphs, as well as the quality of the solution. They chose several graphs, including random and real-life benchmarks to perform the evaluation, and found that in almost all examples they obtain solutions of the same quality and with no overhead. Qu and Potkonjak built the first theoretical framework for analysing watermarking techniques and describe & provide proofs for their techniques.

Myles *et al.* [24] implemented the QP algorithm using register allocation; they found that a stricter embedding criteria are required due to the unpredictability of the colouring of vertices and the fact that one vertex can be used multiple times. Another, major, flaw in the QP algorithm is that it is

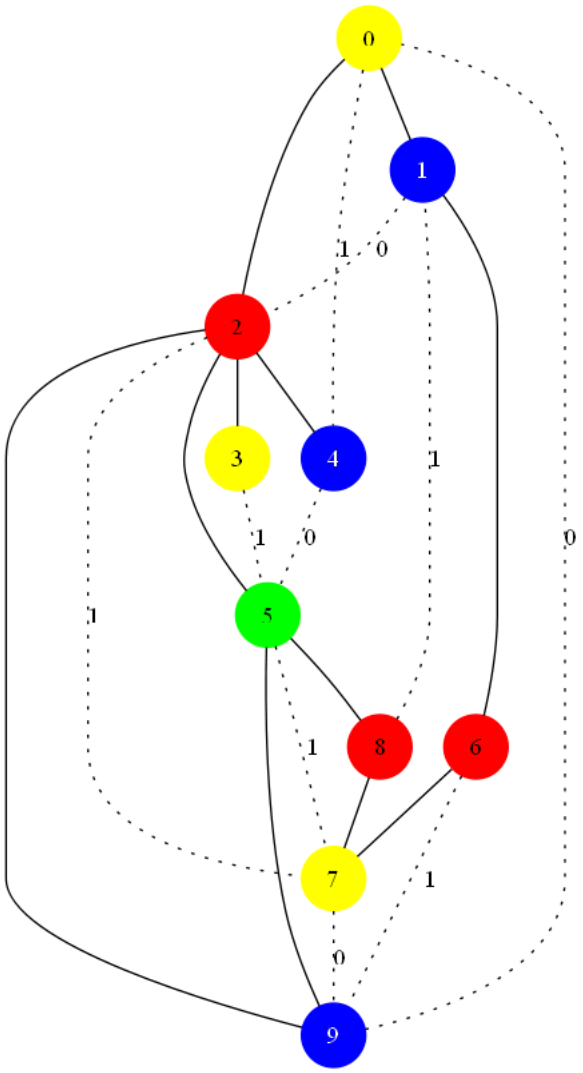

Figure 7: Example interference graph with embedded watermark

possible to insert two different messages into an interference graph and obtained the same watermark graph [22]. The QP algorithm, therefore, is *not extractable* [25], [26].

It has also been shown that the QP graph solution can be modified in such a way that any message could be extracted [27]. We can clearly see this from our example, figure 7: consider the pair (v_3, v_4); we can see that we could deduce that a watermark bit 0 is stored in an edge between these vertices. We know from our embedding that it is not, however it is impossible to tell this without the watermark edges. Qu and Potkonjak dismiss this problem, claiming that it will be hard to build a meaningful message particularly if the original message is encrypted by a one-way function [23].

Due to these flaws in the QP algorithm, Myles *et al.* [24] proposed an improvement which they call the QPS algorithm

Algorithm 2 QP extraction algorithm

Input: a graph $G(V, E)$ with embedded message M
Output: a message $M = m_0 m_1 \ldots$

copy $G(V, E)$ to $G'(V', E')$
for each pair of unconneted vertices v_i, v_j **do**
  n = number of vertices not connected to v_i between v_i and v_j
  if n = 0 **then**
    $M = M + 0$
  else if n = 1 **then**
    $M = M + 1$
  else
    n = number of vertices not connected to v_i between v_j and v_i
    if n = 0 **then**
      $M = M + 0$
    else if n = 1 **then**
      $M = M + 1$
    else
      message bit undefined
    end if
  end if
end for
return M

.

IV. The QPS Algorithm

The key difference between the QP and the QPS algorithms is the selection of vertices. In the QPS algorithm triples of vertices are selected such that they are isolated units that will not effect other vertices in the graph. As watermark bits can only be inserted where there are coloured triples the data-rate of this algorithm is far lower than the QP algorithm.

Definition 4. *Coloured Triple [24]: Given a graph G, a set of three vertices v_1, v_2, v_3 (where $v_1 < v_2 < v_3$) is considered a coloured triple if they are vertices in G, are non-adjacent and they are all the same colour.*

Zhu *et al.* [25] provide clarified versions of the QPS algorithms which eliminate some ambiguities in the originals ; we use these here and they are shown in algorithms 3 and 4.

Figure 8 shows an example graph with a coloured triple {b, c, d}. The embedding algorithm is described, in pseudocode, in algorithm 3. Figure 9 shows the example graph, from figure 6, watermarked with 101_2 using the QPS algorithm. It is clear that the data-rate is much smaller than QPS; we can see that only 3 bits could be embedded using this algorithm.

Myles *et al.* implemented the QPS algorithm in Sandmark [28], an open-source tool for the study of software protection

Algorithm 3 Clarified QPS embedding algorithm

Input: a graph $G(V, E)$, a message $M = m_0 m_1 \ldots$
Output: a graph $G'(V', E')$ with embedded message M
copy $G(V, E)$ to $G'(V', E')$
n = $|V|$ - 1
WV = V
j = 0
for all i from 0 to n **do**
  if possible find the nearest two vertices v_{i1}, v_{i2} in G' such that:
  v_i, v_{i1}, v_{i2} have the same colour
  and are a triple in G' and v_{i1}, $v_{i2} \in WV$.
  WV = WV - $\{v_{i1}, v_{i2}\}$
  j++
  if m_j = 0 **then**
    add edge (v_i, v_{i1}) to E'
  else
    add edge (v_i, v_{i2}) to E'
  end if
end for
return $G'(V', E')$

Algorithm 4 Clarified QPS extraction algorithm

Input: a graph $G'(V', E')$ with embedded message M
Output: a message $M = m_0 m_1 \ldots$
copy $G(V, E)$ to $G'(V', E')$
n = $|V|$ - 1
WV = V
j = 0
for all i from 0 to n **do**
  if possible find the nearest two vertices v_{i1}, v_{i2} in G' such that:
  v_i, v_{i1}, v_{i2} have the same colour
  and are a triple in G' and v_{i1}, $v_{i2} \in WV$.
  WV = WV - $\{v_{i1}, v_{i2}\}$
  j++
  if v_i and v_{i1} have different colours in G' **then**
    m_j = 0
    add edge (v_i, v_{i1}) to E'
  else
    m_j = 1
    add edge (v_i, v_{i2}) to E'
  end if
end for
return $M = m_1, m_2, \ldots m_j$

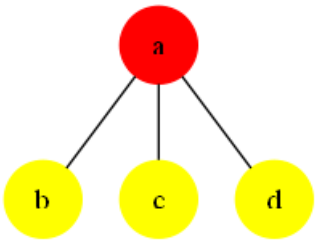

Figure 8: Coloured Triple

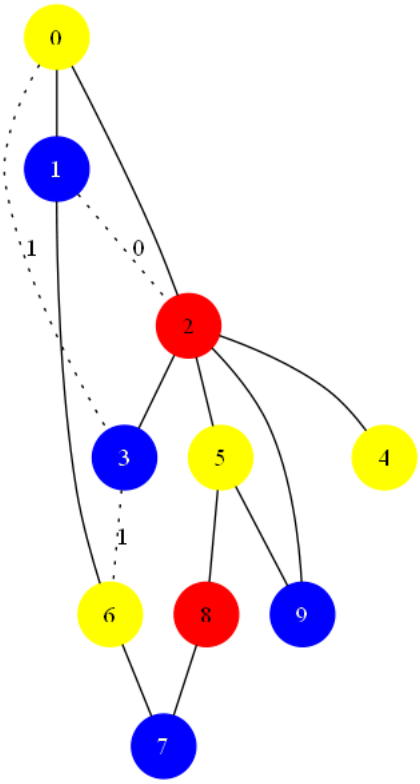

Figure 9: Example interference graph with embedded watermark using the QPS algorithm

algorithms. They performed a variety of empirical tests to evaluate their algorithm's overall effectiveness, examining five properties: credibility, data-rate, stealthiness, part protection and resilience. The results showed that the QPS algorithm has a very low data-rate and is susceptible to a variety of simple attacks, such as obfuscations. However, they also conclude that the QPS algorithm is quite stealthy and is extremely credible. In other words, the watermarks are hard to detect by an attacker whilst readily detectable by the watermark author.

V. The QPI Algorithm

The QPS algorithm is an improvement on the QP algorithm, in terms of extractability, however the QPS algorithm has a very low data-rate. Zhu *et al.* [25] proposed a further

improvement which they call the QPI algorithm. The QPI algorithm changes the definition of the nearest vertices v_{i1}, v_{i2} to v_i and the original QP algorithm used cyclic mod n order for numbers 1,2,...,n, while QPI uses $1 < 2 < \ldots n$.

Definition 5. *Two candidate vertices for the QPI algorithm [25]: for a vertex v_i of a graph G with $|V| = n$ and a colouring of G, v_i has two candidate vertices $v_{i1} \in V$ and $v_{i2} \in V$ if $i < i_1 < i_2 \leq n$ and vertices v_i, v_{i1}, v_{i2} to v_i have the same colour and $(v_i, v_{i2}) \notin E$; furthermore, $\forall j : i < j < i_1$ and $\forall j : i_1 < j < i_2 \leq n$, vertices v_i and v_j have a different colour.*

The QPI embedding and extraction algorithms (see algorithms 5 and 6) uses a new definition of candidate vertices, coloured triples and also changes the vertex colours. Figure 10 shows the example interference graph from figure 6 with an embedded watermark 1011_2. The first watermark bit to embed is 1 and the first vertex is v_0; vertex v_0 has two candidate vertices v_3 and v_4 - these are the same colour and aren't connected to v_0. As with the previous algorithms, we add an edge between v_0 and v_4 because v_4 is the second candidate vertex and the watermark bit is 1. We then change the colour of v_4.

After we change the colour of v_2 in the process of embedding the second bit we leave v_2 without any candidate vertices. We therefore skip vertex v_2 and move on to the next vertex with candidate vertices.

The QPI extraction algorithm requires the original interference graph and the watermarked graph in order to extract the watermark message. We find the candidate vertices from the original graph, then compare the colours in the watermarked graph. For example, vertex v_0 has two candidate vertices v_3 and v_4; v_3 is the same colour in the original and watermarked graph, whereas v_4 is a different colour - the watermark bit is therefore 1.

The QPI algorithm is an improvement on the QPS algorithm with an increased data-rate and it has been shown that QPI algorithm is extractable, unlike the original QP algorithm [25].

VI. The Colour Change Algorithm

The Colour Change (CC) algorithm [29] is another improvement on the QPS algorithm. In the CC algorithm the colouring function is modified to embed a message, rather than modifying the interference graph; the modification only occurs for 1 bits but not 0 bits. The data-rate of the CC algorithm is higher than that of QPS and QPI because each vertex in the interference graph can store 1 watermark bit.

Figure 11 shows the example interference graph (from figure 6) with the watermark 1011011010_2 embedded; for clarity, the vertices are annotated with the watermark bits.

The first step in the CC algorithm is to colour the interference graph to obtain the colouring function γ. We then adapt the colouring function γ to produce a new colouring

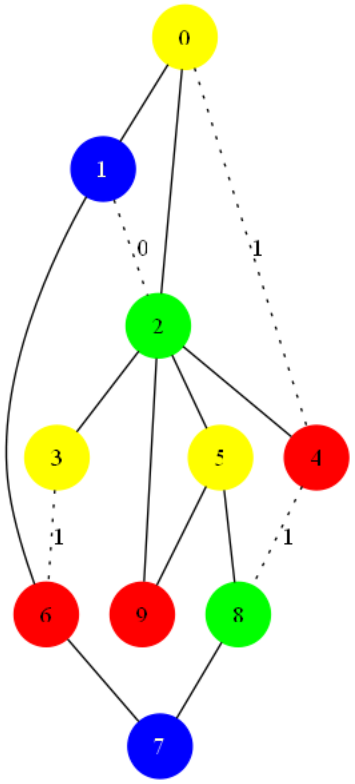

Figure 10: Example interference graph with embedded watermark using the QPI algorithm

Algorithm 5 QPI embedding algorithm

Input: a graph $G(V, E)$ and a message $M = m_0 m_1 \ldots m_k$

Output: a graph G' with embedded message M

```
n = |V| - 1
G' = G
j = 0
for all i from 0 to n do
  if v_i has two candidate vertices v_i1, v_i2 then
    j++
    if w_j = 0 then
      add edge (v_i, v_i1) in G'
      change the colour of v_i1 to a different colour from
      the current colors used in G'
    else
      add edge (v_i, v_i2) in G'
      change the colour of v_i2 to a different colour from
      the current colors used in G'
    end if
  end if
end for
return G'
```

Algorithm 6 QPI extraction algorithm

Input: an unwatermarked graph $G(V, E)$ and watermarked graph $G'(V', E')$
Output: a message M

```
n = |V| - 1
j = 0
for all i from 0 to n do
  if v_i has two candidate vertices v_i1, v_i2 in G then
    j++
    if v_i and v_i1 have different colours in G' then
      m_j = 0
      add edge (v_i, v_i1) in G
      change the colour of v_i1 to a different colour from
      the current colors used in G
    else
      m_j = 1
      add edge (v_i, v_i2) in G
      change the colour of v_i2 to a different colour from
      the current colors used in G
    end if
  end if
end for
return M = m_0 m_1 ... m_j
```

function γ' which includes the embedded watermark (see algorithm 7). For example, the first watermark bit to embed is 1 and the original colouring of vertex v_0 is yellow; we choose the lowest possible legal colour (see table I for colour encodings) to replace red. The only vertices which do not change colour are the vertices in which a 0 bit will be encoded.

Colour	Encoding
Yellow	1
Blue	2
Red	3
Green	4

Table I

In order to extract the watermark, the CC extraction algorithm (see algorithm 8) takes the original interference graph G and the graph colouring function γ generated by the embedding algorithm. The original graph colouring γ' is obtained by colouring G which is then compared with the colours obtained by γ. If the original colouring of a vertex matches the new colouring then the watermark bit is 1; otherwise the watermark bit is 0.

Vertex	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
γ	1	2	2	1	1	1	1	2	3	3
γ'	3	2	4	2	1	2	3	2	1	3

Table II

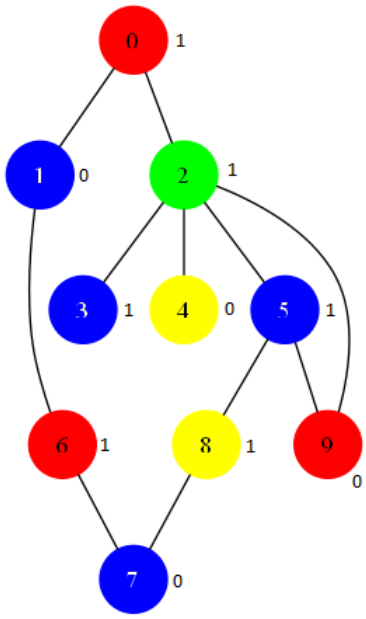

Figure 11: Example interference graph with embedded watermark using the CC algorithm

Table II shows the output of the original colouring function γ and the new colouring function γ' for each of the vertices.

Lee and Kaneko [30] experimentally evaluated their CC algorithm and found that, on average, it takes 0.75 extra colours to embed a message in an interefence graph. They suggest using the CC algorithm for programs which contain many variables, as the data-rate is equal to the number of vertices in the interference graph. They introduce a second algorithm - Color Permutation - for use in programs which require a large number of registers.

VII. The Colour Permutation Algorithm

The Colour Permutation (CP) algorithm [29], [30] uses a similar idea to the CC algorithm, as they both change the colouring function for an intereference graph to encode the watermark bits. The CP algorithm converts the watermark bit string into a natural number M, and then chooses the M^{th} permutation of the lexicographically ordered colours to replace the original colour. The algorithm uses the relationship between the factorial number system and lexicographically ordered permutations [31], [32] to obtain the M^{th} permutation.

The encoding algorithm is show in algorithm 9. The published extraction algorithm is incorrect and we provide a corrected version here (algorithm 10).

The data rate of the CP algorithm is proportional to the number of colours c, given by $log_2 c!$, whereas the data rate of the CC algorithm is equal to the number of variables used. We can therefore only store two watermark bits in our

Algorithm 7 Colour Change embedding algorithm

Input: a graph $G(V, E)$ and a message $M = m_0 m_1 \dots m_k$

Output: a colouring function γ

```
n = |V| - 1
if n < k then
  abort('embedding failed')
else
  γ = ColourGraph(G)
  for j = 0 to n do
    if m_j = 1 then
      find smallest i such that i ≠ γ[j] and i ≠ γ[j'] for
      any (j, j') ∈ E such that j' < j or m_j' = 0
      γ[j] = i
    end if
  end for
end if
return γ
```

Algorithm 8 Colour Change extraction algorithm

Input: a graph $G(V, E)$, graph colouring function γ
Output: message M

```
n = |V| - 1
γ' = ColourGraph(G)
for j = 0 to n do
  if γ[j] = γ'[j] then
    m_j = 0
  else
    m_j = 1
  end if
end for
return M = m_0 m_1 ... m_n
```

example interefence graph (figure 6) as it contains 3 colours ($log_2 3! = 2$).

Figure 12 shows the example interference graph after encoding 10_2 with the CP algorithm. The embedding algorithm first converts 10_2 into the natural number 1; and then converts this to the factoradic $\{1, 0, 0\}$ and obtains the corresponding permutation $\{2, 1, 3\}$. The new colour permutation is applied to the original interference graph, so that yellow and blue colours are swapped.

To extract the embedded watermark we perform the opposite of the embedding algorithm (see algorithm 10). First the factoradic $\{1, 0, 0\}$ is obtained by comparing the original colouring and the new colouring which is then converted to the original natural number.

Table III shows the output of the original colouring function γ and the new colouring function γ' obtained using the CP algorithm.

Lee and Kaneko [30] experimentally evaluated their CP algorithm and found that the embedded watermark is stealthy

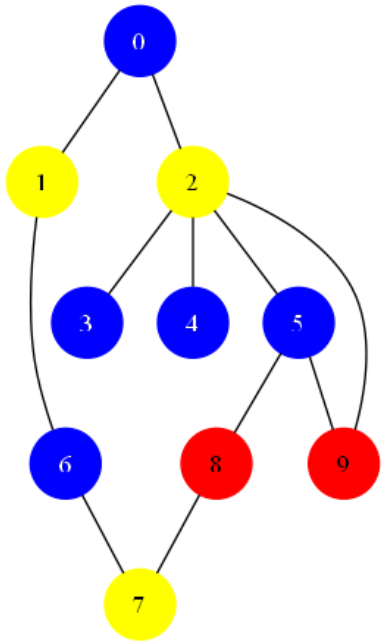

Figure 12: Example interference graph with embedded watermark using the CP algorithm

Vertex	0	1	2	3	4	5	6	7	8	9
γ	1	2	2	1	1	1	1	2	3	3
γ'	2	1	1	2	2	2	2	1	3	3

Table III

and has a high credibility but low-resilience to semantics-preserving attacks.

The CP algorithm has the advantage that it will never introduce a new register; however, this means that the program must already use a large amount of registers in order to embed a large message. The CC and CP algorithms cannot embed a watermark consisting of all zeros.

VIII. The Selected Colour Change Algorithm

Li *et al.* [33] proposed a more efficient algorithm based on the CC algorithm which they call Selected Colour Change (SCC). The SCC algorithm is very similar to the CC algorithm, however, the efficiency increase is obtained by only changing the colours of either the 1 or the 0 bits, but not both. If the occurance of 0 bits is higher than 1 bits in the watermark then 0 bits are changed; otherwise 1 bits are changed. The choice of which bits are changed is stored in a virtual vertex v_{-1} to allow the embedding algorithm to extract the watermark.

Figure 13 shows the example intereference graph (from figure 6) using the SCC algorithm 11 to embed the watermark 1011011010_2. We change the colouring function for the 0 bits because the watermark string contains four 0 bits and six 1 bits; we also set the virtual node v_{-1} to 0 to inform

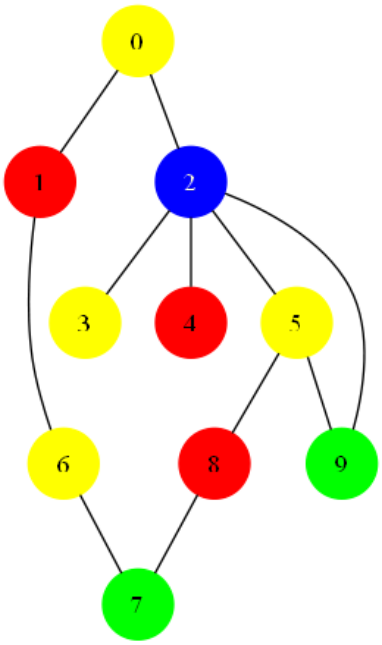

Figure 13: Example interference graph with embedded watermark using the SCC algorithm

the extraction algorithm of this. We change the colouring function in the same manner as the CC algorithm; that is, we choose the lowest possible legal colour as the replacement.

For example, the first bit to embed is 1 therefore we do not change the colouring function for vertex v_0. However, the next bit is a 0 therefore we so we change the colouring function to output a different colour for v_1. We choose red because v_1 is connected to two yellow vertices and v_1 is already blue (see table I for colour encodings). Table IV shows the output of the original colouring function γ and the new colouring function γ' obtained using the CP algorithm.

Vertex	**-1**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
γ	*null*	1	2	2	1	1	1	1	2	3	3
γ'	0	2	1	1	2	2	2	2	1	3	3

Table IV

In order to extract the watermark we observe the value of the virtual vertex v_{-1}. If $\gamma[-1] = 0$ then we assign a 1 to each watermark bit m_j where the original colouring for a vertex j matches the new colouring and a 0 bit if they differ. Otherwise, if $\gamma[-1] = 1$ then we assign a 0 to each watermark bit where the original colouring for a vertex matches the new colouring and a 1 bit if they differ (see algorithm 12).

In our example $gamma[-1] = 0$ so we assign a 1 to each watermark bit m_j where the original colouring for a vertex j matches the new colouring and a 0 bit where they differ.

Li *et al.* [33] evaluate their SCC algorithm and compare the results to the QP, QPS and CC algorithms. They conclude that their SCC algorithm is equivilent to the CC algorithm in many ways, including data-rate, stealthiness and cost. They suggest the SCC algorithm is more efficient than the CC algorithm because it doesn't change all the colours in the interference graph; however, it is not clear how much the efficiency increase affects watermark embedding in real-world programs.

Algorithm 9 Colour Permutation embedding algorithm

Input: a graph $G(V, E)$, a message $M = m_0 m_1 \ldots m_k$
Output: a colouring function γ

```
n = |V| - 1
M = Σ_{i=0}^{k} m_i 2^i
γ = GraphColouring(G)
c = max(γ[0], γ[1], ..., γ[n])
k' = [log_2 c!]
if k' < k then
  abort('embedding failed')
end if
for all j = 1 to c do
  r[j] = M mod (c − j + 1)
  M = M ÷ (c − j + 1)
end for
for all h = 1 to c do
  u[h] = false
end for
for all j = 1 to c do
  cnt = 0
  for all h = 1 to c do
    if u[h] ≠ true then
      if cnt = r[j] then
        u[h] = true
        p[j] = h
        break
      else
        cnt = cnt + 1
      end if
    end if
  end for
end for
for all j = 0 to n do
  γ[j] = p[γ[j]]
end for
return γ
```

IX. Fingerprinting via Register Allocation

Qu *et al.* [34] proposed a modification to the QP algorithm for *fingerprinting*. Fingerprinting is a class of watermarking which embeds a unique identifier into each copy of the software (or music, or films, etc) which allows the origin of the stolen intellectual property to be identified.

Algorithm 10 Corrected Colour Permutation extraction algorithm

Input: a graph $G(V, E)$, graph colouring γ
Output: a message $M = m_0 m_1 \ldots m_k$

```
n = |V| - 1
γ' = ColourGraph(G)
c = max(γ[0], γ[1], ..., γ[n])
for all i = 1 to c do
  p[γ'[i]] = γ[i]
end for
for all i = 1 to c do
  u[i] = false
end for
for all i = 1 to c do
  cnt = 0
  for all j = 1 to c do
    if u[j] ≠ true then
      if j = p[i] then
        r[i] = cnt
        u[j] = true
        break
      end if
      cnt = cnt + 1
    end if
  end for
end for
I = 0
for all j = 1 to c do
  I = I × j + r[c − j + 1]
end for
M = ""
while I > 0 do
  M = M + (I mod 2)
  I = I ÷ 2
end while
return M
```

Algorithm 11 Selected Colour Change embedding algorithm

Input: a graph $G(V, E)$ and a message $M = m_0 m_1 \ldots m_k$

Output: a colouring function γ

```
n = |V| - 1
if n < k then
  abort('embedding failed')
else
  n_0 = number of 0 bits in M
  n_1 = number of 1 bits in M
  γ = ColourGraph(G)
  if n_0 < n_1 then
    γ[−1] = 0
    for j = 0 to n do
      if m_j = 0 then
        find smallest i such that i ≠ γ[j] and i ≠ γ[j']
        for any (j, j') ∈ E such that j' < j or m_j' = 1

        γ[j] = i
      end if
    end for
  else
    γ[−1] = 1
    for j = 0 to n do
      if m_j = 1 then
        find smallest i such that i ≠ γ[j] and i ≠ γ[j']
        for any (j, j') ∈ E such that j' < j or m_j' = 0

        γ[j] = i
      end if
    end for
  end if
end if
return γ
```

The generic approach proposed is to generate n solutions to a graph colouring problem and assign a given solution to one customer only. This approach guarantees that each customer will be able to be uniquely identified by the register allocation generated by their unique inteference graph colouring.

X. QP Algorithms and Public-Key Cryptography

It is advisable to encrypt the watermark message before embedding to prevent an adversery from extracting a watermark, even if they know the extraction algorithm. For example, if an adversery obtains a program which they know contains a watermark embedding with the QPS algorithm they can simply run the QPS extraction algorithm and obtain the watermark; they could then claim that they inserted the watermark. Jian *et al.* [35] propose a technique based on a combination of RSA public-key encryption [36] and the QPI algorithm [25]. Simply, they suggest that the watermark bit string is encrypted before being embedded. If an adversery extracts the encrypted watermark they will not be able to decipher it, if the encryption is strong enough.

XI. Conclusion

The QP algorithm has been shown to be unextractable [22] and an implementation of the QPS algorithm has shown that the algorithm has a low data-rate [37] and is highly suspceptible to semantics-preserving transformation attacks [24]. Any other register allocation based software watermarking algorithm will also be susceptible to semantics-preserving transformations. More specifically, any transformation which alters the interference graph or register allocation will easily remove a watermark.

Register allocation based algorithms maybe be stealthy [24], as they do not add any extra code, but we do not

Algorithm 12 Selected Colour Change extraction algorithm

Input: a graph $G(V, E)$, graph colouring function γ
Output: message M

```
n = |V| - 1
γ' = ColourGraph(G)
if γ[−1] = 0 then
  for j = 0 to n do
    if γ[j] = γ'[j] then
      m_j = 1
    else
      m_j = 0
    end if
  end for
end if
if γ[−1] = 1 then
  for j = 0 to n do
    if γ[j] = γ'[j] then
      m_j = 1
    else
      m_j = 0
    end if
  end for
end if
return M = m_0 m_1 ... m_n
```

recommended them for protecting software due to their low-resilience to attacks. However, academic research continues in this area with the latest register allocation based approach [33] published this year.

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] G. Cronin, "A taxonomy of methods for software piracy prevention," Department of Computer Science, University of Auckland, New Zealand, Tech. Rep., 2002.

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

[3] J. Hamilton and S. Danicic, "An evaluation of current java bytecode decompilers," in *Ninth IEEE International Workshop on Source Code Analysis and Manipulation*, vol. 0. Edmonton, Alberta, Canada: IEEE Computer Society, 2009, pp. 129–136.

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

[5] C. Collberg and C. Thomborson, "Software watermarking: Models and dynamic embeddings," in *Principles of Programming Languages 1999, POPL'99*, Jan. 1999. [Online]. Available: http://www.cs.auckland.ac.nz/collberg/Research/Publications/CollbergThomborsonLow99a/index.html

[6] W. F. Zhu, "Concepts and techniques in software watermarking and obfuscation," PhD Thesis, The University of Auckland, 2007.

[7] W. Zhu, "Informed recognition in software watermarking," in *Proceedings of the 2007 Pacific Asia conference on Intelligence and security informatics*. Chengdu, China: Springer-Verlag, 2007, pp. 257–261.

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

[9] J. Nagra, C. Thomborson, and C. Collberg, "A functional taxonomy for software watermarking," in *Aust. Comput. Sci. Commun.*, M. J. Oudshoorn, Ed. Melbourne, Australia: ACS, 2002, pp. 177–186.

[10] K. HATTANDA and S. ICHIKAWA, "The evaluation of davidsons digital signature scheme," *IEICE TRANS. FUNDAMENTALS*, vol. E87A, no. 1, Jan. 2004.

[11] R. Davidson and N. Myhrvold, "Method and system for generating and auditing a signature for a computer program," Jun. 1996, microsoft Corporation, US Patent 5559884.

[12] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher, Eds. Plenum Press, 1972, p. 85103.

[13] G. J. Chaitin, M. A. Auslander, A. K. Chandra, J. Cocke, M. E. Hopkins, and P. W. Markstein, "Register allocation via coloring," *Computer Languages*, vol. 6, no. 1, pp. 47 – 57, 1981.

[14] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Addison Wesley, Aug. 2006, published: Hardcover.

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

[16] T. Lindholm and F. Yellin, *The Java(TM) Virtual Machine Specification (2nd Edition)*. Prentice Hall PTR, Apr. 1999, published: Paperback.

[17] A. Buckley, E. Rose, A. Coglio, B. S. Corporation, I. Sun Microsystems, I. Tmax Soft, S. Technologies, and E. AG, "JSR 202: JavaTM class file specification update," 2006. [Online]. Available: http://jcp.org/en/jsr/detail?id=202

[18] D. Kramer, B. Joy, and D. Spenhoff, "The java[tm] platform," Sun Microsystems, Tech. Rep., May 1996. [Online]. Available: http://java.sun.com/docs/white/platform/javaplatformTOC.doc.html

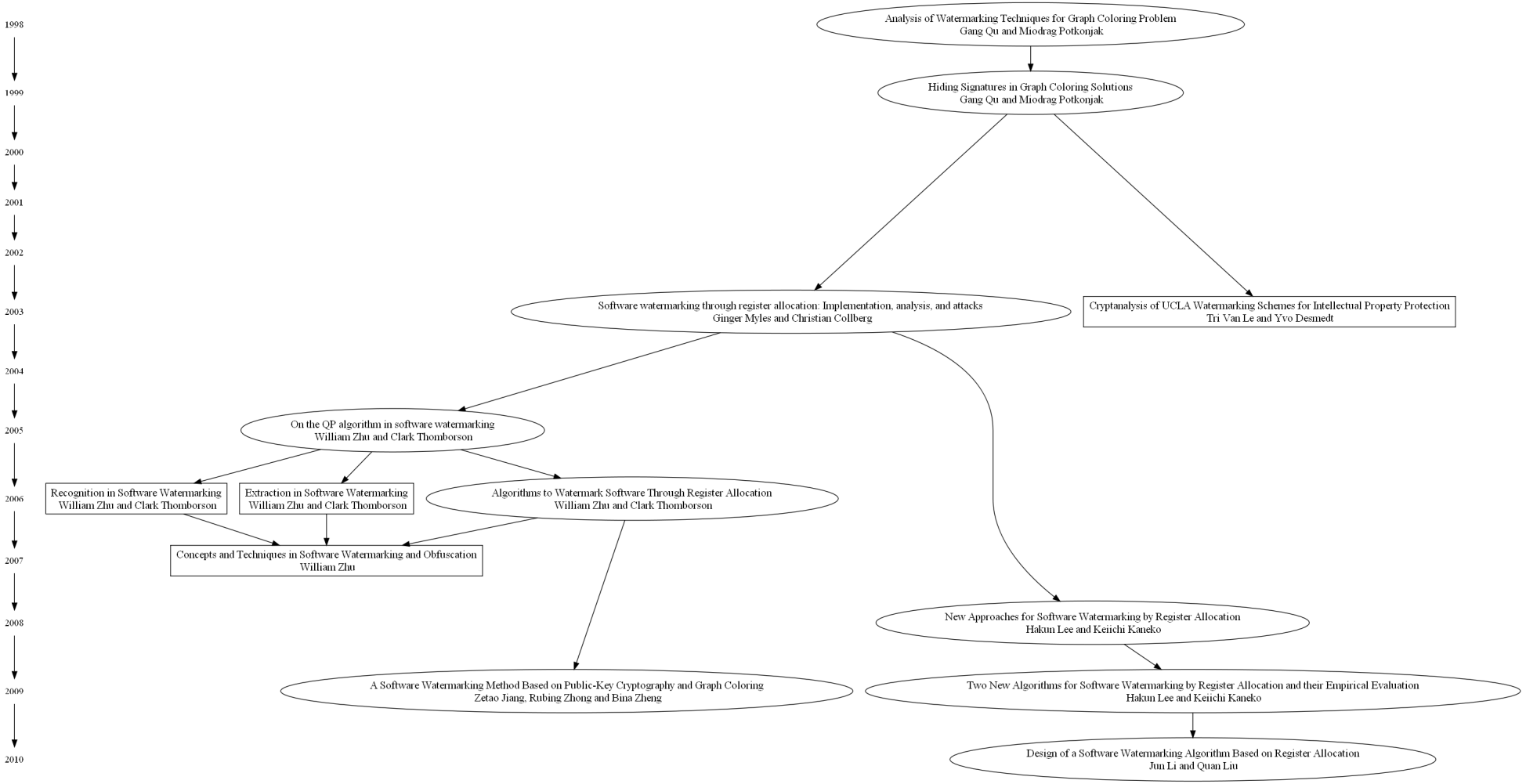

Figure 14: Evolution of Register Allocation Based Software Watermarking. Oval shapes represent new algorithms, while rectangular shapes represent evaluations.

[19] R. Tolksdorf, "Programming languages for the java virtual machine JVM," http://www.is-research.de/info/vmlanguages/index.html, 2010. [Online]. Available: http://www.is-research.de/info/vmlanguages/index.html

[20] K. J. Gough and D. Corney, "Implementing languages other than java on the java virtual machine," 2001.

[21] G. Qu and M. Potkonjak, "Analysis of watermarking techniques for graph coloring problem," in *Proceedings of the 1998 IEEE/ACM international conference on Computer-aided design*. San Jose, California, United States: ACM, 1998, pp. 190–193.

[22] W. Zhu and C. Thomborson, "Extraction in software watermarking." in *MM&Sec*, S. Voloshynovskiy, J. Dittmann, and J. J. Fridrich, Eds. ACM, 2006, pp. 175–181.

[23] G. Qu and M. Potkonjak, "Hiding signatures in graph coloring solutions," in *Information Hiding*, 1999, pp. 348–367.

[24] G. Myles and C. Collberg, "Software watermarking through register allocation: Implementation, analysis, and attacks," in *International Conference on Information Security and Cryptology*, ser. Lecture Notes in Computer Science, vol. 2971/2004. Springer Berlin / Heidelberg, 2003.

[25] W. Zhu and C. Thomborson, "Algorithms to watermark software through register allocation," ser. Lecture notes in computer science, vol. 3919. Berlin, ALLEMAGNE: Springer, 2006, undefined Anglais.

[26] ——, "Recognition in software watermarking," in *Proceedings of the 4th ACM international workshop on Contents protection and security*. Santa Barbara, California, USA: ACM, 2006, pp. 29–36.

[27] T. V. Le and Y. Desmedt, "Cryptanalysis of UCLA watermarking schemes for intellectual property protection," in *Revised Papers from the 5th International Workshop on Information Hiding*. Springer-Verlag, 2003, pp. 213–225.

[28] C. Collberg, "Sandmark," Department of Computer Science, Aug. 2004. [Online]. Available: http://www.cs.arizona.edu/sandmark/

[29] H. Lee and K. Kaneko, "New approaches for software watermarking by register allocation," in *Proceedings of the 2008 Ninth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*. IEEE Computer Society, 2008, pp. 63–68.

[30] ——, "Two new algorithms for software watermarking by register allocation and their empirical evaluation," in *Proceedings of the 2009 Sixth International Conference on Information Technology: New Generations*. IEEE Computer Society, 2009, pp. 217–222.

[31] D. E. Knuth, *Art of Computer Programming, Volume 3: Sorting and Searching (2nd Edition)*, 2nd ed. Addison-Wesley Professional, May 1998, vol. 3, published: Hardcover.

[32] J. Arndt, *Matters Computational: ideas, algorithms, source code*, 1st ed. Springer, Oct. 2010, to be published. Free electronic version online. [Online]. Available: http://www.jjj.de/fxt/#fxtbook

[33] J. Li and Q. Liu, "Design of a software watermarking algorithm based on register allocation," in *e-Business and Information System Security (EBISS), 2010 2nd International Conference on*, 2010, pp. 1–4.

[34] G. Qu and M. Potkonjak, "Fingerprinting intellectual property using constraint-addition," in *Design Automation Conference*, 2000, pp. 587–592, citeseer.nj.nec.com/qu00fingerprinting.html.

[35] Z. Jiang, R. Zhong, and B. Zheng, "A software watermarking method based on Public-Key cryptography and graph coloring," in *Genetic and Evolutionary Computing, 2009. WGEC '09. 3rd International Conference on*, 2009, pp. 433–437. [Online]. Available: http://www.computer.org/portal/web/csdl/doi/10.1109/WGEC.2009.76

[36] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[37] J. Hamilton and S. 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, Oct. 2010, to appear.