Extraction in Software Watermarking

William Zhu[*]
the Department of Computer Sciences,
the University of Auckland,
Auckland, New Zealand
fzhu009@ec.auckland.ac.nz

Clark Thomborson
the Department of Computer Sciences,
the University of Auckland,
Auckland, New Zealand
cthombor@cs.auckland.ac.nz

ABSTRACT

The widespread use of the Internet makes software piracy and unauthorized modification easier and more frequent. Among the many techniques developed for protecting software copyrights is software watermarking which embeds secret messages into software to identify its owners and developers. While digital watermarking for media such as video, audio, and text is a popular research field, software watermarking is still a relatively new scientific area. The key concepts in software watermarking are informal; some are even confusing. Formalizing these fundamental terms would facilitate the research in this field. In this paper, we formally define the following concepts involved in embedding watermarks into and extracting watermarks from a program in software watermarking: embedding, set of candidate watermarks, representative set, representative degree, extracting, extractability, blindly extractability, and representative extracting.

Through the concepts of the representative sets and the representative degree of an embedding algorithm and a program, we characterize the intrinsic property of an extractable embedding algorithm for software watermarking. Furthermore, the concept of the representative extracting algorithm is used to show the best thing we can get for a general software watermarking embedding algorithm.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.0 [**Software Engineering**]: General—*Protection mechanisms*; K.4.1 [**Computers and Society**]: Public Policy Issues—*Intellectual property rights*; K.4.4 [**Computers and Society**]: Electronic Commerce—*Intellectual property; Security*; K.5.1 [**Legal Aspects Of Computing**]: Hardware/ Software Protection—*Copyrights; Proprietary rights*

[*]Corresponding author. Phone: +64-9-3737-599 ext. 82298; fax: +64-9-3737-453;

General Terms

Security

Keywords

Software Watermark, Security, Embed, Extract, Graph, Interference Graph

1. INTRODUCTION

With the rapid development of software industries, the protection of intellectual property of software from piracy takes on greater importance both in computer business and academia. Software watermarking is an approach that embeds a message into software to claim its ownership [2, 3, 5]. It is an effective mechanism to protect the intellectual property of software developers. For more details about software watermarking and related techniques, please refer to [5, 23, 25, 26].

Though several excellent algorithms are available in literatures [1, 3, 9, 10, 11, 12, 14, 15, 16, 18, 19, 20, 22], software watermarking is still a relatively new research field, and some of the core concepts are informal, even confusing. The formalization of them will benefit researchers in this area and facilitate the progress in it. In this paper, we focus on one important concept in software watermarking - extraction of a watermark embedded in software. From the discussion of these terms related to this concept, we can see more clearly the problems to be solved in a good software watermarking system.

This paper is organized as follows. After this introduction is Section 2 which presents a concise literature review of this research field. Section 3 defines the concepts of embedding, the set of candidate watermarks, representative sets and the representative degree. While embedding is the basic concept in software watermarking, the set of candidate watermarks denotes all watermarks that an embedding algorithm can actually insert into a program. The representative sets and the representative degree characterize extractability of software watermarking, an intrinsic property of an embedding algorithm. In Section 4, we define the concepts of extracting, extractability, blind extractability, and informed extractability. Section 5 presents the concepts of representative extracting, which establishes the relationship between the extractable and the representative extractable embedding algorithms. The representative extracting concept contains the important property of a general embedding algorithm. Section 6 has the summary our paper.

2. OVERVIEW

The concepts of software watermarking abound in the research literature [2, 3, 4, 6, 7, 8, 17, 21]. Early detailed definitions of software watermarking concepts appear in the papers [2, 3]. They have descriptions of concepts such as attack types, static watermarks and dynamic watermarks, stealth, resilience, data rate. Authors of those papers categorize dynamic watermarking techniques into three classes: Easter egg watermarks, data structure watermarks, and execution trace watermarks.

The authors of paper [17] introduce four terms in software watermarking from a functional view: authorship mark, fingerprinting mark, validation mark, and licensing mark. They also defines several other concepts such as visible and invisible watermarks, robust and fragile watermarks. The paper [21] provides extensive concepts such as static watermarks and dynamic watermarks. Embedding and extracting are more formally defined in paper [4] as the encoding function and exposition function, respectively. Authors of this paper also considers multiple watermarks in a program. In our papers we regard them as parts of a whole watermark.

Our paper contributes to the field by defining (1) the set of candidate watermarks which includes all watermarks that can be actually embedded into a program by a certain embedding algorithm; (2) defining representative sets and the representative degree which characterize an extractable embedding algorithm; and (3) defining the representative extracting algorithm which shows what is possible for a general embedding algorithm.

3. EMBEDDING

A software watermarking system should do at least two basic things. Firstly, it can embed a watermark into software. Secondly, it can extract all bits of the watermark inserted by itself, or it can judge the existence of the watermark embedded by this system. In this section, we discuss the problem of embedding all bits of a watermark into software.

Definition 1. *(Watermark) A watermark is a message of bits of 0 and 1 with a finite length ≥ 0. The watermark with length 0 is called an empty watermark and it is denoted by ϵ. The length of a watermark W is denoted as $len(W)$.*

We denote the set of watermarks as $\mathcal{W}$. Precisely, $\mathcal{W} = \{0,1\}^{\infty}$.

Concatenation of two watermarks. Let $U = u_1u_2 \dots u_m, V = v_1v_2 \dots v_n \in \mathcal{W}$; the concatenation of U and V is a new watermark $W = u_1u_2 \dots u_m v_1v_2 \dots v_n$ where we add all bits of V after U. U is a prefix of W and V is a suffix of W.

For a finite set S, the cardinal number of S is denoted as $|S|$. For an infinite set S, $|S| = \infty$.

Definition 2. *(Embedding) Let $\mathcal{P}$ denote the set of programs and $\mathcal{W}$ the set of watermarks. We call a function $A : \mathcal{P} \times \mathcal{W} \to \mathcal{P}$ a watermark embedding algorithm, or, for sake of simplicity, an embedding algorithm, or an embedder.*

If $P' = A(P, W)$ for a $P \in \mathcal{P}$ and a $W \in \mathcal{W}$, P' is called a watermarked program. We also call the program $P \in \mathcal{P}$ an original program corresponding to the watermarked program P'.

Example 1. *(The trivial embedder Triv) Define the embedder $Triv : \mathcal{P} \times \mathcal{W} \to \mathcal{P}$ as follows*

$$\forall P \in \mathcal{P} \text{ and } W \in \mathcal{W},\ Triv(P, W) = P$$

This embedder is called the trivial embedder.

Definition 3. *(Normal Embedding) If $A(P, \epsilon) = P$, the embedder A is called normal.*

Example 2. *The trivial embedder is normal.*

The following example involves the QP algorithm which watermarks software through register allocation by coloring the interference graph of this software. It is an important background and example in our paper. To describe the QP algorithm well, we need the following terms and notations.

K-colorable: We say a graph $G = (V(G), E(G))$ is k-colorable if it has an ancillary coloring function $C : V \to \{1, 2, ..., k\}$ with the following properties.

$$\forall (u, v) \in E(G) \implies C(u) \neq C(v)$$

Cyclic mod n ordering: We use "$<_i$" to denote the cyclic mod n ordering relation for a fixed i, such that $i <_i (i{+}1) <_i \dots <_i n <_i 1 <_i \cdots <_i i-1$. Where there is no confusion over the value of i, we omit the subscript in $<_i$.

Potential watermark vertices [18, 19]: For a vertex v_i of a graph G with $|V| = n$, we say $v_{i_1} \in V$ and $v_{i_2} \in V$ are the potential watermark vertices with respect to v_i if $i <_i i_1 <_i i_2$; $(v_i, v_{i_1}) \notin E$; $(v_i, v_{i_2}) \notin E$; $\forall j : i <_i j <_i i_1$, $(v_i, v_j) \in E$; and $\forall j : i_1 <_i j <_i i_2$, $(v_i, v_j) \in E$.

PW and PWV: For every vertex $v_i \in G$, we define a predicate $\mathrm{PW}(v_i, G)$. If there exist two potential watermark vertices with respect to v_i, the value of $\mathrm{PW}(v_i, G) = \mathrm{TRUE}$. Otherwise $\mathrm{PW}(v_i, G) = \mathrm{FALSE}$.

When $\mathrm{PW}(v_i, G) = \mathrm{TRUE}$, the potential watermark vertices with respect to v_i are denoted as $\mathrm{PWV}(v_i, G, 1) = v_{i_1}$ and $\mathrm{PWV}(v_i, G, 2) = v_{i_2}$. When $\mathrm{PW}(v_i, G) = \mathrm{FALSE}$, the values of $\mathrm{PWV}(v_i, G, 1)$ and $\mathrm{PWV}(v_i, G, 2)$ are undefined.

Example 3. *The QP embedding algorithm [18, 19] in Fig. 1 is normal.*

```
Input:  An unwatermarked graph $G$ with $n = |V|$ and
        An unbounded series of message bits:
        $W = w_1 w_2 \dots w_m$
Output:  A watermarked graph $G'$.
Algorithm:
$G' = G$;
$j = 1$;
if m > n       // not all bits of $W$ inserted in $G$
    return $G$
for each $i$ from 1 to $n$
    if j > m       // all bits of $W$ already inserted in $G$
        return $G'$
    if $\mathrm{PW}(v_i, G')$
        $G' = G' + (v_i, \mathrm{PWV}(v_i, G', w_i + 1))$;
        j++;
if m > j       // not all bits of $W$ inserted in $G$
    return $G$
else
     return $G'$
```

Figure 1: A clarified version of the QP algorithm

Definition 4. *(Set of Candidate Watermarks) A watermark $W \in \mathcal{W}$ is called a candidate watermark with respect to a program P and an embedder A if $A(P, W) \neq P$.*

All candidate watermarks constitute the set of candidate watermarks of the program P and the embedder A. This set is denoted as $Candidate(P, A)$.

The set of candidate watermarks of a program P and an embedder are all watermarks that can actually be inserted into P by A. Embedding other watermarks into P will not change the original program.

Example 4. *(Set of candidate watermarks) Let A be the QP algorithm in Fig. 1 and P be a program with the interference graph having 4 vertices v_1, v_2, v_3, v_4 and two edges $(v_1, v_3), (v_2, v_4)$. Then, the set of candidate watermarks of A and P is $\{$0, 1, 00, 01, 10, 11, 010, 011, 110, 111$\}$. The interference graphs of original program and the watermarked programs are in Fig. 2. We can see from Fig. 2 that the interference graph for the watermarked program after a watermark 010 inserted is the same as that after a watermark 111 inserted. For this reason, a watermark embedded by the QP algorithm can not be extracted reliably.*

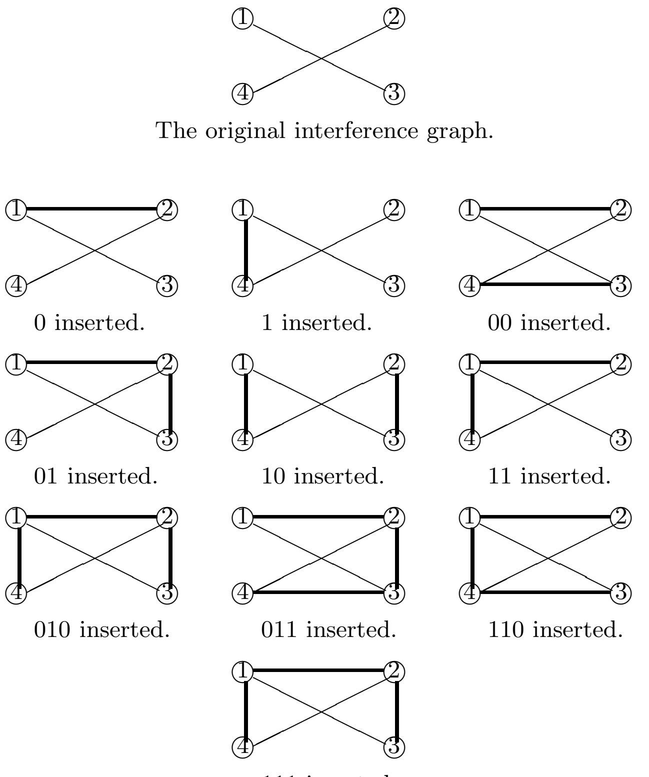

Figure 2: The interference graphs of the original and the watermarked programs

Example 5. *(The set of candidate watermarks of the trivial embedder is empty.) $\forall P \in \mathcal{P}$, $Candidate(P, Triv) = \phi$. The trivial embedder is the only embedder A such that $\forall P \in \mathcal{P}$, $Candidate(G, A) = \phi$.*

For most embedding algorithms of software watermarking, any watermark can be inserted into a program, but, some, especially the QP algorithm, can embed only a limited numbers of watermarks. Without defining the set of candidate watermarks, the following confusing will occur.

In paper [13], Le and Desmedt develop a destroy algorithm to attack the QP embedding algorithm. The main result about this destroy algorithm is as follows (Theorem 2 in [13]).

Let C'' be the output of the destroy algorithm proposed in [13] on input (G, C'), where C' is the output of the QP embedding algorithm [18, 19] on input graph G. Then the verification algorithm [13] will always output yes on input (G, C'', M) for arbitrary signature M.

We must solve the contradiction between the arbitrary M in Theorem 2 in [13] and that, for a graph with n vertices, the QP algorithm can insert at most only n bits of a message into such a graph. The concept of "Set of candidate watermarks" is one of a set with all watermarks embeddable into a program by a certain software watermarking embedding algorithm. If, for example, only a part of a watermark $W = UV$ with $len(V) > 0$, U, can be inserted into a program, we would think we embed U instead W into this program.

Definition 5. *(Finite Embedding) An embedder $A : \mathcal{P} \times \mathcal{W} \to \mathcal{P}$ is called a finite embedder if, for every program P and embedder A, the set of candidate watermarks of P and A is finite.*

Example 6. *The QP algorithm in Fig. 1 and the trivial embedder are finite.*

Definition 6. *(Representative Sets) Let A be an embedder. For a program P, $A(P, W) = A(P, W')$, $W, W' \in \mathcal{W}$ is an equivalent relation in the set of candidate watermarks of P and A. Every equivalent class is called a representative set of the embedder A and the program P.*

All watermarks in a representative set of an embedder A and a program P have the same effect on program P - they generate the same watermarked program by the embedder A.

Property 1. *If A is a normal embedder, then, for any program P, the watermark ϵ does not belong to the set of candidate watermarks of A and P.*

Definition 7. *(Representative Degree) The maximal cardinal number of the representative sets of an embedder A and a program P is called the representative degree of the embedder A and the program P and is denoted as $Repdegree(P, A)$.*

The concept of the representative degree is used to judge the effectiveness of an embedder A. The smaller it is, the better the A is.

Example 7. *(Representative sets and degree) A and P is as in Example 4. The representative sets of A and P are $\{$0$\}$, $\{$1$\}$, $\{$00$\}$, $\{$01$\}$, $\{$10$\}$, $\{$11$\}$, $\{$011$\}$, $\{$110$\}$, $\{$010,111$\}$. So, $Repdegree(P, A) = 2$.*

4. EXTRACTING ALGORITHM AND EXTRACTABILITY

Definition 8. *(Extracting) Let A be an embedder; a function $X : \mathcal{P} \times \mathcal{P} \to \mathcal{W}$ is called an extracting algorithm corresponding to the embedder A if X has the following property: $\forall P, P' \in \mathcal{P}$,*
$X(P', P) = W$, if $W \in Candidate(P, A)$ and $P' = A(P, W)$.
$X(P', P) = \epsilon$, otherwise.

Property 2. *(Normality of Extracting Algorithms) Let A be an embedder; an extracting algorithm corresponding to the embedder A is normal in the sense that $X(P, P) = \epsilon$.*

Proof. By the definition, if $\epsilon \in Candidate(P, A)$, then $P \neq A(P, W)$, so $X(P, P) = \epsilon$.

If $\epsilon \notin Candidate(P, A)$, then $X(P, P) = \epsilon$ by the definition.

DEFINITION 9. *(Extractability) We say watermarks embedded by an embedding algorithm A are* extractable *if there exists an extracting algorithm corresponding to the embedding algorithm A. We also say X demonstrates that the embedding algorithm A is extractable, or more simply, that A is extractable.*

THEOREM 1. *Let A be an embedder. If A is extractable, then, for any program P, any representative set of P and A has only one element. Especially, Repdegree$(P, A) = 1$.*

On the other hand, if, for any program P, Repdegree$(P, A) = 1$, then A is extractable

Proof. The proof of the first part of this theorem is evident from the definitions. Therefore we prove only the second part of this theorem.

An extracting algorithm corresponding to the embedding algorithm A is defined as follows. $\forall P', P \in \mathcal{P}$,

$X(P', P) = W$ if there is a $w \in Candidate(P, A)$ such that $P' = A(P, W)$.

$X(P', P) = \epsilon$, otherwise.

If the representative degree of any program and A is 1, the above function $X : \mathcal{P} \times \mathcal{P} \rightarrow \mathcal{W}$ is well-defined. It is an extracting algorithm corresponding to the embedding algorithm A.

EXAMPLE 8. *(The QP embedding algorithm in Fig. 1 is not extractable.) From Example 7 and Theorem 1, The QP algorithm is not extractable.*

EXAMPLE 9. *(The QPI embedding algorithm [24] in Fig. 3 is extractable.) Because the QP algorithm is not extractable, we develop an improvement on the QP algorithm, the QPI algorithm.*

The extracting algorithm [24] corresponding to the QPI embedding algorithm is in Fig. 4.

```
Input:  an original graph $G(V, E)$ with $n = |V|$
          a message to be embedded into the $G(V, E)$:
          $W = w_1 w_2 \ldots w_m$
Output: a watermarked graph $G'$ with message $W$ embedded
          in it
Algorithm:
copy $G(V, E)$ to $G'(V', E')$
add two vertices $v_{n+1}, v_{n+2}$ to $V'$
$j = 1$;
if m > n      // not all bits of $W$ inserted in $G$
     return $G$
for each $i$ from 1 to $n$
     if j > m      // all bits of $W$ already inserted in $G$
          return $G'$
     if PW$(v_i, G')$
          $G' = G' + (v_i, \mathrm{PWV}(v_i, G', w_i + 1))$;
          j++;
if m > j      // not all bits of $W$ inserted in $G$
     return $G$
return $G'$
```

Figure 3: The QPI Embedding Algorithm

DEFINITION 10. *(Blind and Informed Extractability) For an embedding algorithm $A : \mathcal{P} \times \mathcal{W} \rightarrow \mathcal{P}$, if there exists a*

```
Input:  the original graph $G(V, E)$ with $n = |V|$
          the watermarked graph $G(V', E')$
Output: the message $W$ embedded in the watermarked graph
          $G(V', E')$
Algorithm:
add two vertices $v_{n+1}, v_{n+2}$ to $V$
If $G$ is not a subgraph of $G'$
     return $W = \phi$
j = 0
BITFINISHED = FALSE
for each $i$ from 1 to $n$
     if PW$(v_i, G')$
          if $(v_i, v_{i_1}) \in E'$
               if BITFINISHED == TRUE
                    return $W = \phi$
               j++
               $w_j = 0$
               $G = G + (v_i, \mathrm{PWV}(v_i, G, w_j + 1))$;
          else if $(v_i, v_{i_2}) \in E'$
               if BITFINISHED == TRUE
                    return $W = \phi$
               j++
               $w_j = 1$
               $G = G + (v_i, \mathrm{PWV}(v_i, G, w_j + 1))$
          else          // all possible bits extracted
               BITFINISHED = TRUE
     else if $v_i$ is connected to all other vertices in $G'$ but
               not connected to all other vertices in $G$.
          return $W = \phi$
return the message $W = w_1 w_2 \ldots w_j$
```

Figure 4: The QPI Extraction Algorithm

function $Y : \mathcal{P} \rightarrow \mathcal{W}$ having the following properties: $\forall P' \in \mathcal{P}$,

$Y(P') = W$, if $P' = A(P, W)$ for a $P \in \mathcal{P}$ and a $W \in Candidate(P, A)$.

$Y(P') = \epsilon$, otherwise

then we say that watermarks embedded in programs of $\mathcal{P}$ using an embedding algorithm $A : \mathcal{P} \times \mathcal{W} \rightarrow \mathcal{P}$ are blindly extractable. *Such an X is called a blind extractor for embedder A. If there exists a blind extractor X for an embedder A, we say A is blindly extractable.*

It is easy to construct an extractor $X : \mathcal{P} \times \mathcal{P} \rightarrow \mathcal{W}$ from a blind extractor Y by defining

$X(P', P) = Y(P')$ if $Y(P') \neq \epsilon$ and $P' = A(P, Y(P'))$.

$X(P', P) = \epsilon$ otherwise.

Thus "blindly extractable" implies "extractable".

If there is no blind extractor for an embedder A, but there is an extracting algorithm corresponding to A, A is called an informed embedder*, and X is called an informed extractor for A. The combination (A, X) is called a* informed watermark extraction system*, when X is an informed extractor for A.*

Note: In the above definitions, an informed embedder cannot be a blind embedder, and vice versa.

In the following, we present two examples for blind extractability.

EXAMPLE 10. *(The QPI embedding algorithm in Fig. 3 is not blindly extractable)*

EXAMPLE 11. *(A blindly extractable embedding algorithm) An embedder A is defined as follows:*

For any program P, if $W = 101$, $A(P, W)$ is P plus an extra variable declaration. Otherwise, $A(P, W) = P$. A is blindly extractable. In fact, we have an extracting algorithm X corresponding to A as follows:

For any $P' \in \mathcal{P}$, if P' has at least one variable declaration, $X(P') = 101$. Otherwise, $X(P') = \epsilon$.

5. REPRESENTATIVE EXTRACTING

DEFINITION 11. *(Representative Extracting) Let A be an embedder; a function $X : \mathcal{P} \times \mathcal{P} \to \mathcal{W}$ is called a representative extracting algorithm corresponding to the embedder A if it has the following property:*

$$\forall W \in \mathcal{W} \text{ and } \forall P \in \mathcal{P},$$
$$X(A(P, X(A(P, W), P)), P) = X(A(P, W), P).$$

The background for the representative extracting algorithms is as follows. For some embedding algorithm A, for example, we may get a same watermarked program P' after inserting any watermark in $\{101, 1110, 0010\}$ by A into program P. As a representative extracting algorithm X corresponding to the embedder A, $X(P', P)$ should be one of the watermarks in $\{101, 1110, 0010\}$. In the current software watermarking algorithms available, this phenomenon appears in the QP algorithm.

THEOREM 2. *If X is an extracting algorithm corresponding to an embedder A, then X is also a representative extracting algorithm corresponding to A.*

THEOREM 3. *(Characteristic of representative sets of a representative extracting algorithm) Let A be an embedder. If X is a representative extracting algorithm corresponding to A, then, for any program P and any representative watermark set $R = \{\ldots, W, \ldots\}$ of P and A, $X(A(P, W), P) \in R$.*

THEOREM 4. *For every embedder A, there exists a representative extracting algorithm corresponding to A.*

Proof. For an embedder A, a corresponding representative extracting algorithm X can be defined as follows:
$\forall P, P' \in \mathcal{P}$,
$X(P, P') = W$, if $\exists W \in Candidate(P, A)$ such that $A(P, W) = P'$;
$X(P, P') = \phi$, otherwise.
It is easy to see X is a representative extracting algorithm corresponding to A.
From Theorem 4, we have the following concept.

DEFINITION 12. *(Proper Representative Extractable) Let A be an embedder; it is called proper representative extractable if it is not extractable.*

THEOREM 5. *An embedder A is proper representative extractable if and only if there exists a program P such that $Repdegree(P, A) > 1$.*

EXAMPLE 12. *(A representative extracting algorithm corresponding to the QP algorithm in Fig. 1) The algorithm in Fig. 5 is a representative extracting algorithm corresponding to the QP embedding algorithm. The extracting algorithm outlined in [18] is really not an extracting algorithm but a representative extracting algorithm. For the same program in Example 4, if 010 is inserted, this representative extracting algorithm will get 010, but, if 111 is inserted, this representative extracting algorithm will get 010, not 111.*

Generally, we can develop several other representative extracting algorithms for a proper representative extractable embedder. For example, another representative extracting algorithm for the QP algorithm is in Fig. 6. For the same program in Example 4, if 010 is inserted, this representative extracting algorithm will get 111, not 010, but, if 111 is inserted, this representative extracting algorithm will get 111.

```
Input:  an unwatermarked graph $G(V, E)$ with $n = |V|$
          a watermarked graph $G'$
Output:  a message $W$ embedded in $G'$
Algorithm:
j=0
BITFINISHED = FALSE
for each $i$ from 1 to $n$
    if $\mathrm{PW}(v_i, G)$
        $G = G + (v_i, \mathrm{PWV}(v_i, G, w_i + 1))$;
        if $(v_i, v_{i_1}) \in G'$
            if BITFINISHED == TRUE
                return $W = \phi$
            j++
            $w_j = 0$
        else if $(v_i, v_{i_2}) \in G'$
            if BITFINISHED == TRUE
                return $W = \phi$
            j++
            $w_j = 1$
        else         // all bits extracted
            BITFINISHED = TRUE
    else if $v_i$ is connected to all other vertices in $G'$ but
            not connected to all other vertices in $G$.
        return $W = \phi$
return $W = w_1 w_2 \ldots w_j$
```

Figure 5: A representative extracting algorithm for QP algorithm

EXAMPLE 13. *(The QP algorithm in Fig. 1 is proper representative extractable) Example 8 shows the QP algorithm is not extractable and Example 12 shows the existence of a representative extracting algorithm corresponding to the QP algorithm, so the QP algorithm is proper representative extractable.*

DEFINITION 13. *(Informed and Blind Representative Extracting) Let A be an embedder; if function $Y : \mathcal{P} \to \mathcal{W}$ satisfies*

$$\forall W \in \mathcal{W} \text{ and } \forall P \in \mathcal{P},$$
$$Y(A(P, Y(A(P, W)))) = Y(A(P, W)).$$

then Y is called a blind representative extracting algorithm corresponding to the embedding algorithm A.

A representative extracting algorithm corresponding to the embedding algorithm A but is not a blind representative extracting algorithm corresponding to the embedding algorithm A is called an informed representative extracting algorithm corresponding to the embedding algorithm A.

DEFINITION 14. *(Blindly Representative Extractable) Let A be an embedder; if there is a blind representative extracting algorithm corresponding to the embedding algorithm A, we say watermarks embedded by algorithm A are* blindly representative extractable, *or, simply, A* blindly representative extractable. *The combination (A, X) is called a* blind representative watermark extraction system.

THEOREM 6. *If an embedding algorithm A is blindly extractable, then it is also blindly representative extractable.*

```
Input:  an unwatermarked graph $G(V,E)$ with $n = |V|$
          a watermarked graph $G'$
Output:  a message $W$ embedded in $G'$
Algorithm:
If $G$ is not a subgraph of $G'$
     return $W = \phi$
j=0
BITFINISHED = FALSE
for each $i$ from 1 to $n$
     if $\mathrm{PW}(v_i, G)$
          $G = G + (v_i, \mathrm{PWV}(v_i, G, w_i + 1))$;
          if $(v_i, v_{i_2}) \in G'$
               if BITFINISHED == TRUE
                    return $W = \phi$
               j++
               $w_j = 1$
          else if $(v_i, v_{i_1}) \in G'$
               if BITFINISHED == TRUE
                    return $W = \phi$
               j++
               $w_j = 0$
          else         // all bits extracted
               BITFINISHED = TRUE
     else if $v_i$ is connected to all other vertices in $G'$ but
               not connected to all other vertices in $G$.
          return $W = \phi$
return $W = w_1 w_2 \ldots w_j$
```

Figure 6: Another representative extracting algorithm for QP algorithm

THEOREM 7. *If an embedding algorithm is proper representative extractable, then the representative degree of some program and A is greater than 1.*

EXAMPLE 14. *(The QP algorithm in Fig. 1 is not blindly representative extractable)*

EXAMPLE 15. *(A blindly representative extractable embedder) An embedder A is defined as follows:*

For any program P, if $W = 101$ or if $W = 110$, $A(P,W)$ is P plus an extra variable declaration. Otherwise, $A(P,W) = P$. A is blindly extractable. In fact, we have a blind extracting algorithm Y corresponding to A as follows.

For any $P' \in \mathcal{P}$, if P' has at least one variable declaration, $Y(P') = 101$. Otherwise, $Y(P') = \epsilon$.

6. CONCLUSIONS

Algorithmic design, even with an adequate formal statement of the problem to be solved, is an art, not a science. Without a precise statement of the problem, we cannot hope to prove the correctness of any algorithm, and indeed we may have difficulty even explaining what the algorithm is intended to do.

When we started this research project, we thought that it would be a simple matter to prove the QP algorithm either correct or incorrect. However we could not do this until we devised appropriate definitions for two basic problems in watermarking: recognition and extraction. None of our initial, intuitively-formed, problem definitions were sufficient to support a careful analysis of the QP algorithm; and we found little support for a careful analysis in the published literature. However we did succeed in devising a serviceable set of definitions, allowing us to complete a careful analysis of the QP algorithm and its variants. In the process we discovered some subtle bugs and algorithmic issues. Our major findings are summarized very briefly below.

We use the concepts of representative sets and representative degree to characterize the extractable embedding algorithm. We define the concept of the representative extracting algorithm to show the intrinsic property of a general embedding algorithm. We also define the blindly extractable embedding algorithm and informedly extractable embedding algorithm, as well as the blindly representative embedding algorithm and informedly representative embedding algorithm. This is the first appearance of the concepts of the set of candidate watermarks, the representative sets, the representative degree, and the representative extracting algorithm are first appeared in published literature. Informed and blind extractability are also introduced in this paper.

The recognition of watermarks in software is another important work for our next paper.

7. ACKNOWLEDGMENTS

Authors are in part supported by the New Economy Research Fund of New Zealand. The first author also thanks the Department of Computer Sciences at the University of Auckland for its financial support for his research.

8. REFERENCES

[1] G. Arboit. A method for watermarking java programs via opaque predicates. In *The Fifth International Conference on Electronic Commerce Research (ICECR-5)*, 2002.

[2] C. Collberg, C. Thomborson, and D. Low. On the limits of software watermarking. In *Technical Report #164, Department of Computer Science, The University of Auckland*, 1998.

[3] C. Collberg and C. Thomborson. Software watermarking: Models and dynamic embeddings. In *Proceedings of Symposium on Principles of Programming Languages, POPL'99*, pages 311–324, 1999.

[4] C. Collberg, S. Jha, D. Tomko, and H. Wang. Uwstego: A general architecture for software watermarking. *Technical Report TR04–11*, Aug. 31 2001.

[5] C. Collberg and C. Thomborson. Watermarking, tamper-proofing, and obfuscation - tools for software protection. *IEEE Transactions on Software Engineering*, 28:735–746, Aug. 2002.

[6] C. Collberg, S. Kobourov, E. Carter, and C. Thomborson. Error-correcting graphs for software watermarking. In *29th Workshop on Graph Theoretic Concepts in Computer Science*, July 2003.

[7] C. Collberg, G. Myles, and A. Huntwork. Sandmark–a tool for software protection research. *IEEE Security and Privacy*, 1(4):40–49, 2003.

[8] C. Collberg, C. Thomborson, and G. Townsend. Dynamic path-based software watermarking. In *Computer Science Department, University of Arizona (USA), Technical report*, volume TR04–08, April 2004.

[9] C. Collberg, E. Carter, S. Debray, A. Huntwork, C. Linn, and M. Stepp. Dynamic path-based software watermarking. In *SIGPLAN '04 Conference on Programming Language Design and Implementation*, June 2004.

[10] P. Cousot and R. Cousot. An abstract interpretation-based framework for software watermarking. In *Principles of Programming Languages 2003, POPL'03*, pages 311–324, 2003.

[11] D. Curran, N. Hurley, and M. O. Cinneide. Securing java through software watermarking. In *Proceedings of the 2nd international conference on Principles and practice of programming in Java*, pages 311–324, 2003.

[12] O. Esparza, M. Fernandez, M. Soriano, J. Munoz, and J. Forne. Mobile agent watermarking and fingerprinting tracing malicious hosts. In *LNCS 2736*, pages 927–936, 2003.

[13] T. Le and Y. Desmedt. Cryptanalysis of ucla watermarking schemes for intellectual property protections. In *LNCS 2578*, pages 213–225. Springer-Verlag, 2003.

[14] A. Monden, H. Iida, and K. ichi Matsumoto. A practical method for watermarking java programs. In *The 24th Computer Software and Applications Conference*, pages 191–197, 2000.

[15] G. Myles and C. Collberg. Software watermarking through register allocation: Implementation, analysis, and attacks. In *LNCS 2971*, pages 274–293, 2004.

[16] J. Nagra and C. Thomborson. Threading software watermarks. In *IH'04*, 2004.

[17] J. Nagra, C. Thomborson, and C. Collberg. Software watermarking: Protective terminology. In *Proceedings of the ACSC 2002*, 2002.

[18] G. Qu and M. Potkonjak. Analysis of watermarking techniques for graph coloring problem. In *IEEE/ACM International Conference on Computer Aided Design '98*, pages 190–193, 1998.

[19] G. Qu and M. Potkonjak. Hiding signatures in graph coloring solutions. In *Information Hiding Workshop '99*, pages 348–367, 1999.

[20] T. Sahoo and C. Collberg. Software watermarking in the frequency domain: Implementation, analysis, and attacks. In *Computer Science Department, University of Arizona (USA), Technical report*, volume TR04–07, March 2004.

[21] C. Thomborson, J. Nagra, Somaraju, and Y. He. Tamper-proofing software watermarks. In *Proc. Second Australasian Information Security Workshop(AISW2004)*, pages 27–36, 2004.

[22] R. Venkatesan, V. Vazirani, and S. Sinha. A graph theoretic approach to software watermarking. In *4th International Information Hiding Workshop*, Pittsburgh, PA, 2001.

[23] W. Zhu, C. Thomborson, and F.-Y. Wang. A survey of software watermarking. In *IEEE ISI 2005*, volume 3495 of *LNCS*, pages 454–458, May 2005.

[24] W. Zhu and C. Thomborson. Algorithms to watermark software through register allocation. In *DRMTICS 2005*, volume 3919 of *LNCS*, pages 180–191, October 2005.

[25] W. Zhu, C. Thomborson, and F.-Y. Wang. Application of homomorphic function to software obfuscation. In *WISI 2006*, volume 3917 of *LNCS*, pages 152–153, April 2006.

[26] W. Zhu, C. Thomborson, and F.-Y. Wang. Obfuscate arrays by homomorphic functions. In *Special Session on Data Security and Privacy in IEEE GrC 2006, to appear*, pages 770–773, May 2006.