Concepts and Techniques in Software Watermarking and Obfuscation

William Feng Zhu

August 2007

Supervisor: Prof. Clark Thomborson

A thesis submitted in partial fulfillment of
the requirements of Doctor of Philosophy
in Computer Science

The Department of Computer Sciences
The University of Auckland
New Zealand

ABSTRACT

With the rapid development of the internet, copying a digital document is so easy and economically affordable that digital piracy is rampant. As a result, software protection has become a vital issue in current computer industry and a hot research topic.

Software watermarking and obfuscation are techniques to protect software from unauthorized access, modification, and tampering. While software watermarking tries to insert a secret message called software watermark into the software program as evidence of ownership, software obfuscation translates software into a semantically-equivalent one that is hard for attackers to analyze. In this thesis, firstly, we present a survey of software watermarking and obfuscation. Then we formalize two important concepts in software watermarking: extraction and recognition and we use a concrete software watermarking algorithm to illustrate issues in these two concepts. We develop a technique called the homomorphic functions through residue numbers to obfuscate variables and data structures in software programs. Lastly, we explore the complexity issues in software watermarking and obfuscation.

Acknowledgment

It has taken nine years to officially complete my doctoral study. This is a hard experience, but it is also a fruitful period. I want to use this opportunity to thank all people who helped me in this way or that way.

I would like to thank my supervisor, Professor Clark Thomborson, for valuable guidance and financial support. I have learnt a lot from him. I would also like to thank the advisors for my doctoral study at the University of Auckland – Professor Fei-Yue Wang, Professor Christian Calude, and Doctor Michael Dinneen. The Department of Computer Science also financially supported my doctoral study at Auckland. It deserves my many thanks.

I was fortunate to be part of a very active research group here at Auckland. I have worked closely with and learnt a lot from several students, postdoctorals and faculty members in our group.

Especially, I would like to thank my friends. Mr. Fred He and his wife for their strong support for my study. Dr. Mingkuan Liu also helped me a lot. Thanks also to other friends who encouraged me and helped me.

I would also like to express my gratitude towards my daughter and my family. Their support always makes me overcome every difficulty.

I am also indebted to the following people for their help in my academic career: Huacan He, Dehuang Chen, Jiarui Wu, Huaxiao Zhang, Cheng Ge, Yiyu Yao,

Xindong Wu, Daniel Zeng, Yixin Zhong, T. Y. Lin, Qing Liu, Jingtao Yao, Guoyin Wang, Weizhi Wu, Guilong Liu, Min Xiao, Zhongjin Cheng, Rong Su, Xuekong Yang, Jianxin Feng, Guoliang Dai, Yizhong Zhan, Fangcai Liu, Ning Lu, and Jishou Ruan.

To My Dear Daughter

Miss Ruolin Zhu

Contents

Chapter 1

Introduction

With the rapid development of the Internet, copying a digital document is so easy and economically affordable that digital piracy is rampant in the world [65]. According to a report [16] of the Business Software Alliance (BSA) and IDC in 2003, world software piracy resulted in lost revenue of nearly 30 billion dollars. The piracy rate is estimated to be as high as 92 percent in some countries. As a result, software protection has become a vital issue in current computer industry and a hot research topic [45, 46]. This dissertation focuses on software watermarking and obfuscation, two interconnected techniques of software protection.

1.1 Software Security and Protection

According to Main and Oorschot [77], computer security can be classified as the following three types.

1. Data security, which is concerned with the confidentiality and integrity of data in transit and storage.
2. Network security, which aims to protect network resources, devices, and services.
3. Software security, which protects software from unauthorized access, modifica-

tion, and tampering.

Computer attacks can be divided into the following three categories [77]:

1. Network threat: Applications such as browsers and mail clients are vulnerable to remote external attack.

2. Insider threat: In this model, attackers have some privileges on either the network or hardware for the applications.

3. Untrusted host threat: In this setting, applications are subject to attacks from the operating system, kernel, and other application systems on the untrusted host machine.

Legal measures and technical approaches exist for software protection. Legal protection has become increasingly important since more products of software are distributed without a signed license agreement. Legal measures are laws concerning copyright, patent, registration and license. Software copyright protects the exclusive rights of a software developer to reproduce or copy, adapt, distribute and publicly perform the work. Generally, copyright laws protect the form of expression of an idea, but not the idea itself. With respect to software, this typically means that it protects a computer program but not the methods and algorithms within the program. Thus, source code and object code are protected against literal copying. While software copyright protects only the expression of an idea in software, software patent laws protect the underlying idea and features of software. Even independent reinvention of the same technique by others does not give them the right to use it. The protection of software by registration has long been an accepted convention. In some countries, registration initially secures legal rights and later use in the country maintains them. A software license is a contract between a developer and a user of computer software. It gives the user the privilege to use software in accordance with the conditions of the license. That privilege might be revoked by the producer at any time, with or without cause.

While legal protection in a country generally can not be extended to other nations and obtaining patent protection for software is relatively expensive, software producers and developers still seek technical measures to protect their software. Technical approaches can be classified as hardware-based methods and software-based methods.

A dongle is a typical hardware-based method. It is a small hardware device that plugs into the serial or USB port of a computer to ensure that only authorized users can use certain software applications. Dongles are only used with expensive, high-end software programs such as accounting and inventory management applications and CAD systems. When a program that comes with a dongle starts, it firstly checks the dongle for verification. If it does not find the dongle, the program quits. Currently, this is the most reliable method of protecting software and is an approach of preventing commercial software of high price from piracy.

The current software-based methods are code authentication, server side execution, code encoding, software watermarking, and software obfuscation. Code authentication is efficient when authentication data are sent through network, but users have complete code, which in theory can be mangled, thus authentication procedures can be removed. For server side execution, software developer does not send final code to users, but provide users the service of the software through executing whole or part of the software on a remote server. It can be used only in presence of high availability of broadband networks. Code encoding protects against tampering of programs and is used very often. The main drawback of this technique is that decoder can be written and used as a universal tool. Software watermarking tries to insert a secret message, called software watermark, into software as the evidence of ownership of it [30, 181]. Software obfuscation translates software into a semantically-equivalent one that is hard for attacker to analyze [33].

1.2 Software Watermarking and Obfuscation

For the first time, Davidson and Myhrvold presented a published software watermarking algorithm in their patent [40]. The early works on software watermarking include paper [52] and patents [85, 123], but the concepts in these works are preliminary and informal. For the first time, Collberg et al. presented detailed definitions for software watermarking [30, 31]. Since then, several new software watermarking algorithms have been proposed [28, 92, 137].

As pioneers in software obfuscation, Collberg, Thomborson et al. explored this research area in paper [28, 32, 33, 75]. These works include a detailed discussion on definitions, problems, techniques, and criterion in software obfuscation. Based on that analysis of alias in a program is an NP-hard problem, Wang et al. [138, 140] presented a software obfuscation method through global arrays and pointers. Their techniques apply to programs written in a programming language such as C which has pointers. For the first time, Barak et al. studied software obfuscation based on a formal cryptographic model called virtual black box in paper [10]. They presented several important impossibility results about software obfuscation. But there still exist positive results for obfuscating point functions with a random oracle in paper [76, 141].

1.3 Goal, Structure and Contribution

In this thesis, our goal is not to construct highly secured software watermarking and obfuscation because this is an extremely difficult or even impossible problem as discussed in Section 2.8. One of our goals is to provide a sound mathematical basis for defining the fundamental process of software watermarking, namely, extraction and recognition. As for software obfuscation, our goal is to correct an important

obfuscation algorithm by Chow et al., and show how to apply it to the obfuscation of integer arrays.

The main content of this thesis is divided into two parts: Software watermarking and software obfuscation.

Part 1 is devoted to discussing software watermarking. Chapter 2 is a detailed survey of software watermarking. It overviews the problems, taxonomy, and current techniques in this research subject. Chapter 3 discusses one of important concepts in software watermarking, extraction. We formalize this essential concept in this chapter. In this discussion, we develop a software watermarking with linear complexity. Chapter 4 formalizes another important concept in software watermarking, recognition. It is a tricky concept in software watermarking.

Part 2 focuses on software obfuscation. Chapter 5 is a brief survey of software obfuscation. Chapter 6 is a detailed theory of homomorphic function and its application to obfuscating variables. This method is based on residue number encoding. Further applications of homomorphic function to obfuscating data structures are presented in Chapter 7.

We address applications of rough sets theory to security issues, present some topics for future research, and conclude in part 3.

In two appendixes, we list the publications and academic activities of the author of this thesis during his doctoral study in the Department of Computer Sciences at the University of Auckland.

The main contributions of this thesis are as follows:

1. Formalization of extraction and recognition, two essential concepts in software watermarking.

2. Development of a software watermarking algorithm.

3. Establishment of a technique called homomorphic function to obfuscate variables and data structures.

The following issues involved in this thesis deserves further study:

1. Formalizing concepts of software watermarking from an information point of view [86, 87, 88].

2. The security of homomorphic functions in software obfuscation.

3. Application of rough set theory to software watermarking and obfuscation.

Part I

Software Watermarking

Chapter 2

Survey of Software Watermarking

Software watermarking is a method of software protection by embedding secret information into the text of software. We insert such secret information to claim ownership of the software. This enables the copyright holders to establish the ownership of the software by extracting this secret message from an unauthorized copy of this software when an unauthorized use of this software occurs.

Software watermarking can be regarded as a branch of digital watermarking, which started about 1954 [36]. Since the publication of a seminal work by Tanaka et al. in 1990 [131], digital watermarking has made considerable progress and become a popular technique for copyright protection of multimedia information. Research on software watermarking started in the 1990s. The patent by Davidson and Myhrvold [40] presented the first published software watermarking algorithm. The preliminary concepts of software watermarking also appeared in paper [52] and patents [85, 123]. Collberg et al. presented detailed definitions for software watermarking [30, 31]. Unlike other fields of digital watermarking such as multimedia watermarking, software watermarking has not received sufficient attention yet.

Authors of papers [156, 181] have given brief surveys of software watermarking

research. This chapter presents an in-depth look at the state of the art of software watermarking. From Section 2.1 to Section 2.6, we detail the current research status of software watermarking – the taxonomy of software watermarks and software watermark attack models. In Section 2.7, we give a detailed description of the software watermarking algorithms currently available: basic block reordering algorithms, register allocation algorithms, spread-spectrum algorithms, opaque predicate algorithms, threading algorithm, abstract interpretation algorithm, metamorphic algorithm, dynamic path algorithm, and graph-based algorithms. We especially focus on the CT algorithm [25, 30] for software watermarking and the constant encoding technique [55, 133] for protecting this watermarking.

2.1 Overview of Software Watermarking

Among the techniques that can protect software from piracy, software watermarking [53, 31] is unique in that it does not aim to prevent software piracy from happening, but instead aim to show evidence of a piracy event. Multimedia watermarking serves a similar purpose in defeating media piracy, such as protecting the copyright of movies in DVD format. It is already a popular research topic in computer science. Software watermarking is still a relatively new area and we believe it deserves more attention. Though the goals of multimedia watermarking and software watermarking are similar in that they insert some extra information into digitally-encoding objects, their methodologies differ. In software watermarking, when a watermark is embedded into a program, the operating semantics of the program must be preserved. In most multimedia watermarking, there is no underlying operating sematics layer in which to embed the watermark. Instead, the only possible "place" to hide a watermark is in the appearance of the multimedia object. In software watermarking, such appearance-modifying watermarks are possible and are called "Easter Eggs". However there are

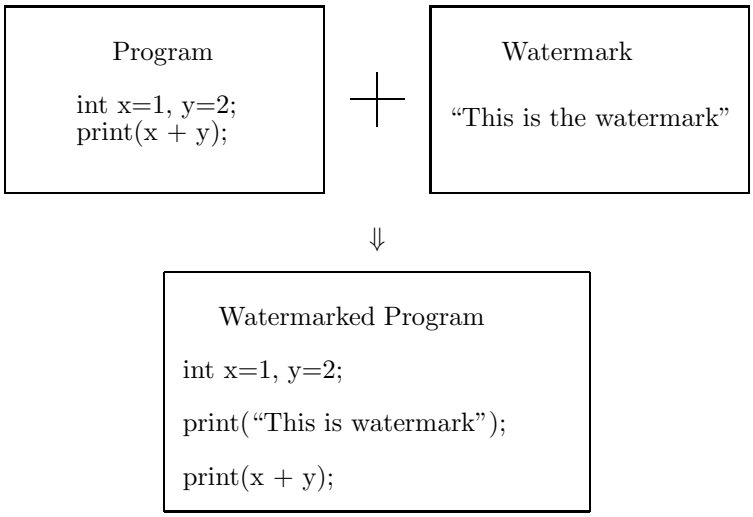


Figure 2.1: Watermark embedding process.

many other possibilities as well.

Software watermarking inserts a piece of information into the program of software. More precisely, let **P** denote the set of programs we want to watermark and **W** the set of watermarks. Software watermarking embeds a w in **W** into a program P in **P** and gets a watermarked program P'. In Fig 2.1, we insert a string “This is the watermark” into the program as a software watermark. In Fig.2.2, we show a simple process to detect a watermark in a program.

2.2 Taxonomy of Software Watermark

We can classify software watermarks in different ways by their functions or properties. The following are some classification schemes from published literature.

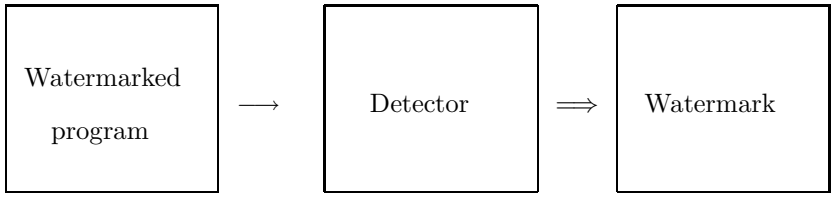


Figure 2.2: Watermark detection process.

2.2.1 Classification by Purpose

Software watermarks can be classified by their functional goals [93, 94]. Each watermark has a single goal in this taxonomy.

1. Prevention marks: Watermarks to prevent unauthorized uses of software.
2. Assertion marks: Watermarks to make a public claim to ownership of software.
3. Permission marks: Watermarks to allow a (limited) change or copy operation to the software.
4. Affirmation marks: Watermarks to ensure an end-user of the software's authenticity.

2.2.2 Classification by Extracting Technique

Classification of software watermark by the extracting technique falls into two classes: static or dynamic [55]. A static software watermark is one inserted in the data area or the text of codes. The extraction of such watermarks needs not run the software. Generally, there are two types of static watermarks [55]: data watermarks and code watermarks. A data watermark is inserted directly into the data area of a program, while a code watermark is inserted into the code area of a program. A simple code watermark involves a permutation of the order of some instructions in a

program. A dynamic software watermark is one inserted in the execution state of a software object. More precisely, in dynamic software watermarking, what has been embedded is not the watermark itself but some codes which cause the watermark to be expressed, or extracted, when the software is run. An example is the dynamic data structure watermark proposed by Collberg and Thomborson [30]. Dynamic software watermarks come in three types: dynamic Easter Egg watermarks, dynamic execution trace watermarks, and dynamic data structure watermarks [30].

2.2.3 Robust and Fragile Software Watermark

Robust software watermarks can be extracted even if it has been subjected to adversarial or casual semantics-preserving or near-semantics-preserving code translation. Such watermarks are used in systems that prevent unauthorized uses (prevention), and in systems that make public claims to software ownership (assertion) [94].

Fragile software watermarks will always be destroyed when the software has been changed. Such watermarks are used in integrity verification of software (affirmations) and in systems that allow limited change and copy (permission) [94].

2.2.4 Visible and Invisible Software Watermark

According to the features that a user of software can experience, software watermarks can be categorized as visible software watermarks and invisible software watermarks. In visible software watermark, some special input will make software generate a legible image like a logo, etc. to show the existence of such visible watermarks in the software (assertion) or to assume a user of authenticity (affirmation). In contrast, invisible software watermarks will not appear as a legible image to the end-user but can be extracted by some algorithm not in the end-user's direct control. These are used in permissions and preventions.

2.2.5 Blind and Informed Software Watermark

According to whether the original program and the watermark are the input to the watermark extractor, software watermarks can be categorized as either blind or informed. [24]. With blind software watermarks, the extractor is given only the watermarked program and the watermark key as its input. In informed software watermark, besides the watermarked program, the extractor is also given the unwatermarked program, or the watermark that was inserted, or both as its input.

2.2.6 Tamperproofing Software Watermark

Tamperproof software watermarks can be extracted even when a skilled adversary purposefully tampered them.

2.3 Attacks on Software Watermark

Attacks on software occur in two ways – malicious client attacks or malicious host attacks. Generally, software watermarking is intended to protect software from attacks by malicious hosts. There are four main ways to attack watermark in software, described as additive attacks, subtractive attacks, distortive attacks, and recognition attacks.

In *additive attacks* adversaries embed a new watermark into the watermarked software, so the original copyright owners of the software cannot prove their ownership from their original watermark.

With *subtractive attacks*, adversaries remove the watermark of the watermarked software without affecting the functionality of the watermarked software.

Adversaries' goal in *distortive attacks* is to modify watermark to prevent it from being extracted by the copyright owners and still keep the usability of the software.

In *recognition attacks*, adversaries modify or disable the watermark detector, or its inputs, so that it gives a misleading result. For example, an adversary may assert that his watermark detector is the one that should be used to prove ownership in a courtroom test.

There are also four common techniques for attacking watermarks [55], called reverse engineering, source code analysis, execution trace analysis, and stack and heap analysis, respectively.

When we want to analyze software, but we can not get access to the source code or the document of the software, we use *reverse engineering* to figure out the software's features or functions. We can also use reverse engineering to find watermark in software.

In contrast, we generally use *source code analysis* to find bugs in software. However, adversaries can also use it in reverse engineering, for example, to find watermarks in software.

Execution trace analysis gives us an execution history, such as function entries and exits, branch points and decisions. Attackers can use such information to understand the software and find watermarks.

Lastly, *stack and heap analysis* involves analyzing the stack space and heap space consumed when running software so that adversaries may find the watermark inserted by a dynamic software watermarking algorithm.

2.4 Protection of Software Watermark

The main types of protection techniques for software watermark are obfuscation and tamperproofing [55]. In *obfuscation*, there occurs a semantics-preserving translation of a program into another program which is hard for an adversary to understand and so it is hard for them to attack it. Obfuscation of a program also makes it hard

for an adversary to locate the watermark. *Tamperproofing* is, likewise, a semantics-preserving translation of a program into another program. It differs from obfuscation in that it is hard for an adversary to modify the new program without changing its behaviors. Thus, even if adversaries locate the watermark inserted into a tamperproved program, it may be hard to remove it without affecting the program's usability.

2.5 Evaluation Criteria of Software Watermark

Roughly speaking, people use four criteria for evaluating the quality of software watermarking [27]. The criterion of *data rate* involves the ratio of the size of the watermark to that of the watermarked program. *Resilience* is the ability to resist against semantics-preserving translations, whereas *stealth* concerns a lack of statistically-distinct visible features between the unwatermarked program and the watermarked program. Finally, *performance criterion* is the ratio of the size and the execution time of the watermarked program to that of the original program.

2.6 Research Platforms for Software Watermarking

The following four systems dominate research platforms for software watermarking: JavaWiz [100], Hydan [44], UWStego [29], and SandMark [26].

JavaWiz is a software watermarking system developed at Purdue University. It can watermark Java source programs, and it is written entirely in Java. This system has implemented the CT algorithm.

Hydan is a software watermarking system developed at Columbia University. It is used to watermark an executable.

UWStego is a software watermarking research tool developed at the University of Wisconsin for experimenting and testing various software watermarking techniques

and has a toolset for developing new software watermarking algorithms.

Lastly, *SandMark* is a comprehensive research tool for software watermarking and obfuscation developed at the University of Arizona. It can be used to measure the effectiveness of software watermarking algorithms. Like JavaWiz, this platform is written in Java.

2.7 Software Watermarking Algorithms

We will describe in this section the major software watermarking algorithms currently available: (1) basic block reordering algorithms, (2) register allocation algorithms, (3) spread-spectrum algorithms, (4) opaque predicate algorithms, (5) threading algorithm, (6) abstract interpretation algorithm, (7) metamorphic algorithm, (8) dynamic path algorithms, (9) mobile agent watermarking, (10) graph-based algorithms, and (11) birthmarks.

2.7.1 Basic Block Reordering Algorithms

In 1996, Davidson and Myhrvold [40] published the first software watermarking algorithm. It embeds a watermark into a program by reordering the basic blocks of the program. In a program, a basic block is a set of sequential instructions with a single entry point and a single exit point. The Davidson-Myhrvold algorithm first chooses a group of basic blocks in an executable, then reorders them to form a watermark with some special feature. This reordering needs to maintain the original flow of execution so that the function of the original program is unchanged. Checking the order of this group of blocks enables us to extract the inserted watermark.

It is easy to attack software watermarks inserted by this algorithm; if we use this algorithm to watermark the watermarked program again, the original watermark will completely be destroyed. This algorithm can be enhanced by using opaque predicate

to establish false dependencies among basic blocks, making it difficult to remove them.

2.7.2 Register Allocation Algorithms

Qu and Potkonjak [115, 116] developed some techniques to watermark solutions to constraint problems such as the graph-coloring problem. The QP algorithm is one of them. It aims to watermark solutions to the graph-coloring problems to protect their intellectual properties. The graph-coloring problem concerns allocating as fewer colors as possible to the vertices of a graph so that no vertices connected by an edge in the graph receive the same color.

Myles and Collberg implemented the QP algorithm for the first time for software watermarking through register allocation [89]. They pointed out that the QP algorithm has a serious flaw since it does not permit reliable recognition. Myles and Collberg proposed a new version of the QP algorithm, the QPS algorithm, which allows robust extraction of watermark in the absence of attacks. Unfortunately, after extensive evaluations, they concluded that the QP algorithm, even the QPS algorithm, is unsuitable for software watermarking of architecture-neutral codes in the presence of determined attackers.

Zhu and Thomborson [180] discussed certain misunderstandings in the QP and QPS algorithms and determine the unextractability of the QP and QPS algorithm through examples. They went on to propose an improvement for the QP algorithm and introduced some potential topics for further research.

2.7.3 Spread-spectrum Algorithms

The spread-spectrum watermarking method was originally developed for watermarking digital media [36]. It represents the data of a document as a vector and modifies each component of the vector with a small random amount. This small amount is

called a watermark. Such watermarks can be recognized by correlation with the extracted watermark signal. The spread-spectrum software watermarking procedure consists of these three steps: representation extraction, watermark insertion, and watermark testing.

Stern et al. [128] proposed the SHKQ algorithm to apply the spread-spectrum watermarking method to software watermarking. In the SHKQ algorithm, code is viewed not as a set of sequential instructions, but as a statistical object. What it really marks is the frequency counts of sets of consecutive instructions. It extracts a group of representation of the code by the Vector Extraction Paradigm proposed by Stern et al., and then applies the spread-spectrum techniques to insert watermarks. We present a simple example of the SHKQ algorithm in Fig. 2.3.

Sahoo and Collberg [120] have implemented the SHKQ algorithm in the software watermarking research tool SandMark. They introduced method overloading to increase the frequency of patterns in cases where code insertion and code substitution are not enough to achieve an acceptably strong watermark signal. Furthermore, they experimented with various attacks on this algorithm.

Curran et al. [37] proposed a spread-spectrum software watermarking method which uses call graph depth as a signal. In this algorithm, at first, a vector from a running program is extracted. The call graph depth is measured at distinct points during the execution of the program to be watermarked on certain particular input. In the end, the program code is modified so that its call graph depth is changed, such that it expresses the watermark when this input is given to the program.

2.7.4 Opaque Predicate Algorithms

An opaque predicate is a predicate whose outcome is known to the system or person who introduces this predicate and the code. Its value must be difficult to analyse by

Original Program

```
 1.  ...
 2.  add X, Y
 3.  ...
     ............
10.  add Y, Z
     ............
36.  add X, Z
     ............
65.  add X, Y
     ............
100.  ...
```

Frequency of the instruction "Add"

4%

Watermarked Program

```
 1.  ...
 2.  add X, Y
 3.  ...
     ............
10.  add Y, Z
     ............
36.  add X, Z
     ............
65.  add X, Y
     ............
78.  add W, Z
     ............
109.  add W, Z
120.  ...
```

Frequency of the instruction"Add"

5%

The value "5%" is the watermark embedded into this program; it can be statistically recognizable as a distinction from an expected value in 4% for "ADD" instruction in a typical program. Many additive statistically-recognizable features, other than the frequency of the "ADD", may be used in a typical application.

Figure 2.3: A simple example of the frequency of patterns in a program as a watermark

automated static analysis. It is a popular technique for software watermarking and obfuscation.

Monden et al. [82, 83] proposed a method to insert watermark into a dummy method never intended for execution. This dummy method is guarded by an opaque predicate. This watermarking algorithm consists of these three phases – dummy method injection, compilation, and stealth watermark injection

With *dummy method injection*, firstly, a dummy method is appended to the unwatermarked Java source program. Then, a dummy method invocation is added to the source program by an unsatisfied opaque predicate as in the following example.

```
if(opaque predicate)    Dummy_Method();
```

In contrast, *compilation* turns a source code into class files. Lastly, *stealth watermark injection* inserts watermarks into the dummy method by writing a bit sequence into the dummy method, by overwriting numerical operands and replacing opcodes. The resulting code is "nonsense" which might be detectable through careful statistical analysis.

Fukushima and Sakurai [48] improved the above algorithm by Monden et al. in a new algorithm. It tries to make the relations in a class file unclear by distributing methods between class files and destroying abstractions. The basic techniques involved in this algorithm are publication of fields, publication of methods, change of methods, distribution of methods, and change of arguments.

Arboit [7] also developed an algorithm to add watermarks into an opaque predicate or an opaque predicate and its associated dummy method, so this algorithm is another improvement to the above algorithm by Monden et al. Myles and Collberg [90] have implemented these types of algorithms on the testbed of SandMark.

2.7.5 The Threading Algorithm

Nagra and Thomborson [92] proposed a threading software watermarking algorithm and implemented it for Java bytecode. In this study we call it the NT algorithm. This algorithm takes advantage of the intrinsic randomness for a thread to run in a multithreaded program. Since it is very hard to analyze such a program, this algorithm claims resilience. In the NT algorithm, the process of embedding watermarks is divided into two steps. Firstly, creating multiple threads of execution – the number of possible execution paths through the program is increased. Inserting suitable locks in certain positions maintains the semantics of the old program. Secondly, locks are added to ensure that only a small subset of the possible paths are actually executed by the watermarked program. The watermark is embedded in those execution paths.

2.7.6 The Abstract Interpretation Algorithm

Cousot and Cousot [35] devised the abstract interpretation algorithm to embed the watermark in values assigned to designated integer local variables during program execution. These values can be determined by analyzing the program under an abstract interpretation framework, enabling the watermark to be detected even if only part of the watermarked program is present.

2.7.7 The Metamorphic Algorithm

In this algorithm, Thaker [132] applied metamorphic code transformations used by metamorphic computer viruses to increase the diversity of software and embed a fingerprint into the software. The pattern for the watermark to be inserted into software is fairly simple, so it is easy to attack this watermarking method.

2.7.8 Dynamic Path Algorithm

Collberg et al. [24] proposed the dynamic path-based software watermarking which inserts watermarks in the runtime branch structure of a program to be watermarked. This algorithm is based on the observation that the branch structure is an essential part of a program and that it is difficult to analyze the branch structure completely because it captures so much of the semantics of the program. The implementation of this algorithm has three stages. In the first one, the tracing stage, we determine the dynamic behaviors of the unwatermarked program by tracing its execution path on a particular input sequence. Then suitable points to insert the watermark must be found. Next, in the embedding stage, modifying the sequence of branches taken and not taken, on the secret input sequence embeds the watermark into the program. Lastly, during extracting stage, we trace the program again using the secret input sequence and check the branch sequence to extract the watermark.

2.7.9 The Mobile Agent Watermarking

Esparza and Fernandez, et al. [47] studied mobile agent watermarking, which uses software watermarking techniques in mobile agents. It aims to guarantee the integrity of the execution of mobile agents. Watermark is embedded into a mobile agent, then it is transferred to the agent's results during its execution. When the agent returns to the origin host, the results of all hosts involved are verified in order to assure the integrity of the execution of the mobile agent.

2.7.10 Graph-based Algorithms

For graph-based algorithms, the watermark is encoded into a graph G in some special kind of graphs. There are two types of graph-based software watermarking algorithms available – the VVS algorithm which is a static one, and the CT algorithm, a dynamic

one.

Venkatesan, Vazirani and Sinha [137] proposed the first graph-based software watermarking, called the VVS algorithm. It is a static software watermarking algorithm. In the VVS algorithm, the graph is the control-flow graph of a program. The VVS algorithm embeds a watermark into a program by adding a constructed control-flow graph to the original control-flow graph of the program to be watermarked. The added control-flow graph is the watermark.

The first dynamic graph algorithm, the CT algorithm, was proposed by Collberg and Thomborson [30], and is one of the strongest software watermarking algorithms [38]. The CT algorithm embeds the watermark in a graph data structure which is built during the execution of the program, and thus is a dynamic software watermarking algorithm. Its extraction process is in Fig. 2.4.

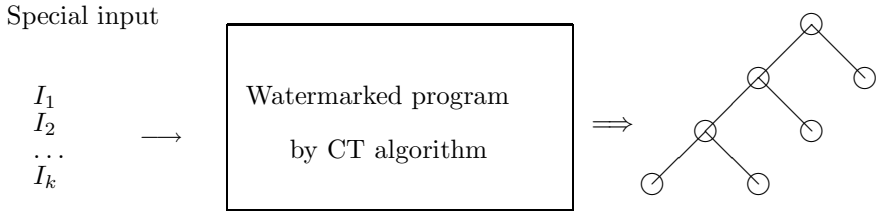


Figure 2.4: Extraction process in the CT algorithm.

There are three basic embedding steps of the CT algorithm. Firstly, a suitable graph G to represent the watermark to be embedded is chosen. Then G is partitioned into several subgraphs. Lastly, the CT algorithm constructs a set of graph-generating code for each above subgraph and inserts those code along a special execution path that is taken when some secret key is provided to the program. If the watermark graph G is well chosen, the watermark embedded by the CT algorithm is stealthy.

Generally, the watermark graph G should not differ from the graph data structures built by real programs. Important conditions are that the maximum out-degree of G should not exceed two or three, and that the graph G have a unique root node so the program can reach other nodes from the root node.

The CT algorithm has two advantages over other software watermarking algorithms. First, the graph data structures embedded in an application program are likely to fit in (be "stealthily") with the original codes, and second, when combined with a suitable graph, it has some error-correcting properties.

On the other hand, the CT algorithm has a disadvantage in that there is no dependency between the graph generating codes and the original codes. Thus, an adversary who can recognize the graph-generating codes accurately can remove them without damaging or affecting program behaviours.

A variant of the above CT algorithm is the CTNSH algorithm [55, 133] proposed by Thomborson et al. The CTNSH algorithm tries to transform some constants in the program text into function calls and to establish some dependencies of the values of these functions on the watermark data structures. It establishes the dependencies of constants in the original program on some codes which are similar to watermark generating code in the CT algorithm. It is a good method for tamper-proofing software watermarks.

Palsberg et al. [100] implemented and evaluated this algorithm for the first time. SandMark has also implemented this algorithm.

2.7.11 Birthmarks

A software birthmark is similar to a software watermark, but different in that software watermarks are certain features inserted purposely into a program while software birthmarks are some characteristics a program inherently owns. Software birthmarks

and software watermarks can be combined to protect each other.

Monden et al. [131] proposed a concept of Java birthmark as a unique set of characteristics of a Java class file for detecting software theft. It is based on the assumption that if a class file has the same birthmark as another class file, these two class files may have the same source, i. e., one is a copy of another. Four types of static software birthmarks were discussed in paper [131] – constant values in field variables, the sequence of method calls, the inheritance structure, and used classes.

Monden et al. proposed two types of dynamic software birthmark for Windows applications [131]. One birthmark is called the sequence of API function calls birthmark, and the other is called the frequency of API function calls birthmark. They are based on the assumption that the API function calls cannot be easily replaced by other instructions, so these birthmarks are expected to be robust.

Myles and Collberg [91] proposed a dynamic software birthmark according to the whole path of a program [67] which represents the dynamic control flow of a program. The specified two important properties of a software birthmark – credibility and robustness. For credibility, the detector should not generate false positives. With robustness, the birthmark should be resilient to sematic preserving transformations. In paper [91], Myles and Collberg had also evaluated the four static software birthmarks in [131].

2.8 Complexity Problems in Software Watermarking

It is ideal to watermark software by some one-way software watermarking algorithms. A one-way function is a function that is easy to compute but hard to invert. In this situation, the watermark inserted by such a one-way software watermarking algorithm is almost totally secured.

One-way functions involve complexity of computation. Complexity theory [34,

124] deals with amount of resources such as time, memory, et al. needed to solve computational problems that we encounter. Computability theory, a similar research area, studies whether a problem can be solved at all. But they are not the same. Computability theory does not care about the resources required.

Computational complexity is mostly interested in lower bound of the resources that are required to solve a certain problem, but we are not interested in the exact amount of the resources needed. Actually, we are interested in the asymptotic complexity, the least asymptotic amount of the resources required to solve the problem.

Computational complexity studies the feasibility of an algorithm. Generally, if an algorithm is feasible, then it runs in polynomial time. Precisely, there is some polynomial p such that the algorithm runs in time at most $p(n)$ on inputs of length n. The words "easy to compute" in a one-way function mean such a function is a feasible one.

In some cases, computational complexity is concerned with infeasible problems. An infeasible problem means it asks impossibly large resources to be solved, even on instances of moderate size. The words "hard to invert" in a one-way function mean there exist no feasible for its inverse, that is, computing its inverse is an infeasible problem. Computational complexity also investigates the relations between different computational problems and between different modes of computation.

Algorithms and problems are categorized into complexity classes. In this thesis, our concern is with the time complexity. We have the time complexities of an algorithm and a problem. The time complexity of an algorithm is the number of steps that this algorithm takes as a function of the size of the input. The time complexity of a problem is the number of steps that it takes to solve an instance of this problem as a function of the size of the input by using the most efficient algorithm. For example, if an instance of a problem that is n bits long and can be solved in n^3 steps, this problem is said to have a time complexity of n^3.

The class of problems that can be solved by a deterministic machine in polynomial time is denoted as P. The class of problems that can be solved by a non-deterministic machine in polynomial time is denoted as NP. A famous open question in complexity theory is whether $P = NP$.

Since the topic of this thesis is on software watermarking and obfuscation, we will not attempt a general discussion of complexity theory here. Lang and Dittmann did an excellent job in evaluating complexity of five different audio watermarking algorithms [66], but little such work exists in software watermarking and obfuscation yet. Recently, Barak et al. did good theoretical work in this area. They defined formally a software obfuscation as follows [10]:

An obfuscator Ob is a feasible compiler that takes as input a program P and produces a new program $Ob(Pr)$ satisfying the following two conditions:

- Functionality: $Ob(Pr)$ computes the same function as Pr.
- Virtual black box property: Anything that can be efficiently computed from $Ob(Pr)$ can be efficiently computed given oracle access to Pr.

An important result proved by Barak et al. is that such an obfuscator is impossible. Another important result is Theorem 3.8 in that paper which concludes that if one-way functions exist, then circuit obfuscators do not exist. Despite the impossibility of existing a universal obfuscator in Barak's model, there are still some positive results. Lynn et al. showed it is possible to obfuscate point functions with a random oracle [76]. Wee [141] provided a simple construction of efficient obfuscators for point functions for a slightly relaxed notion of obfuscation, for which obfuscating general circuits is nonetheless impossible.

As for software watermarking, Barak et al. also had an important conclusion about the existence of one-way software watermarking algorithm. Theorem 8.2 in paper [10] says if one-way functions exist, then no watermarking scheme exists in the sense of

Definition 8.1 in paper [10]. They also proved one-way watermarking functions do not exist in the sense of Definition 8.1 in paper [10]. Until now, we have not seen any discussion about the complexity of modifying a watermark in a program in software watermarking literature. This is still an open question in software watermarking.

If we consider the one-way function from cryptographical point of view, the CT algorithm with key is in a sense a one-way algorithm. If an attacker does not know the key, he cannot easily extract the watermark inserted by the CT algorithm. One approach taken by an attacker might be to use a distinctive feature of the CT algorithm, such as its special data structures used for watermarking to extract all possible watermarks. One-way function involves totally security in theory, but in real world, "adequate security" in watermarking is still of value. For example, it would be of practical interest if we could construct a software watermarking system that would resist a wide range of attacks for a year.

2.9 Conclusion

Software piracy is a worldwide issue and becomes more and more important for software developers and vendors. Software watermarking is one of the many mechanisms to protect the copyright of software. It is a relatively new research field and deserves more attention. Among available software watermarking algorithms, the CT algorithm is one of the most promising methods.

In this part, our goal is not to develop highly secured software watermarking and obfuscation as defined in Barak et al.'s model. We focus on efficiency of software watermark embedding algorithms and the extractability of such algorithms. We are interested in constructing watermark embedding algorithms of linear complexity. In addition, when an embedding algorithm is not extractable, i. e., the inverse of such an algorithm does not exist, we explore whether it is possible to recognise the wa-

termark. Our formal treatment makes it clear that recognition is an easier problem than extraction, in the following sense: a watermark that is not extractable may still be recognizable. We believe that our formalization of the core concepts in software watermarking help researchers in this field find new directions and better techniques for software watermarking. However, it is a task harder than it seems.

In addition, we hope the techniques of software watermarking can be applied to watermarking natural language [8], watermarking relational databases [2, 3], and security informatics [119].

Chapter 3

Extraction

As we said in the previous chapter, there is a need to formalize the core concepts of software watermarking. The focus of this chapter is on formalization of extraction, an important concept in software watermarking. We also define embedding and other concepts related to embedding and extraction.

This chapter is organized as follows. In Section 3.1, we review the literature of the basic concepts in software watermarking. In Section 3.2, we define the concepts of embedding, the set of candidate watermarks, representative sets, and the representative degree. The set of candidate watermarks is used to denote all watermarks which can actually be inserted into a program by an embedding algorithm. The representative sets and the representative degree are used to characterize the intrinsic property of an embedding algorithm that watermarks inserted by such an algorithm are extractable or not. Section 3.3 concerns definitions of extracting, extractability, blind extractability, and informed extractability. Section 3.4 has definitions of the representative extracting and establishes the relationship between the extractable and the representative extractable embedding algorithms. The concept of the representative extracting catches the important property of a general embedding algorithm. In Sec-

tion 3.5, we describe a model for a software watermarking embedding and extraction system based on the concepts and algorithms in this chapter. Section 3.6 is the summary of our chapter.

3.1 Overview

The early detailed definitions of software watermarking concepts appear in the papers [30, 32]. They detail the concepts such as attack types, static watermarks and dynamic watermarks, stealth, resilience, data rate. In these papers, dynamic watermarking techniques are divided into three classes – Easter egg watermarks, data structure watermarks, and execution trace watermarks.

Nagra, Thomborson, and Collberg [93] did an excellent job of defining some concepts in software watermarking. They introduce four terms in software watermarking from a functional view: authorship mark, fingerprinting mark, validation mark, and licensing mark. They also define several other concepts such as visible and invisible watermarks, robust and fragile watermarks.

Collberg, Jha, Tomko, and Wang [29] defined embedding and extracting more formally and these concepts are called the encoding function and exposition function. They considered multiple watermarks in a software. In our thesis we regard them as parts of a whole watermark.

The main contributions of this chapter are fivefold. Firstly, we present the definition of the set of candidate watermarks which includes all watermarks that can be actually embedded into a program by a certain embedding algorithm. In addition, we give the definitions of representative sets and the representative degree which characterize an extractable embedding algorithm. Thirdly, we have the definition of the representative extracting algorithm which shows what we can do for a general embedding algorithm. In this chapter we also use these definitions to develop extracting

and representative extracting algorithms for software watermarking, and lastly, these definitions enable us to design a prototype model for a software watermarking system. These results have been published in paper [172].

3.2 Embedding

In a software watermarking system, we at least should do 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 $\mathbf{W}$*. Precisely,* $\mathbf{W} = \{0,1\}^{\infty}$*.*

Concatenation of two watermarks: Let $U = u_1u_2 \ldots u_m, V = v_1v_2 \ldots v_n \in \mathbf{W}$*, the concatenation of* U *and* V *is a new watermark* $W = u_1u_2 \ldots u_mv_1v_2 \ldots v_n$ *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, $|S| = \infty$.

In practical application, we may restrict in a subset of the watermark set defined above.

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

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

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

$$For\ P \in \mathbf{P}\ and\ W \in \mathbf{W},\ Triv(P, W) = P$$

This embedder is called the trivial embedder.

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

Firstly, we introduce the interference graph of programs which is essential for register allocation in compiling.

Definition 3 *(Interference graph) The interference graph is used for assigning registers to temporary variables in a program. If two variables in the program do not interfere, these two variables have no edge between them. We can use the same register for these two temporary variables. If they interfere in the program, there is an edge between them. In this situation, we are not allowed to use the same register for these two temporary variables.*

Example 2 *A program with the interference graph shown in Figure 3.1.*

$v_1 := 1$

$v_2 := 2$

$v_1 := v_1 + v_2$

$v_2 := 3$

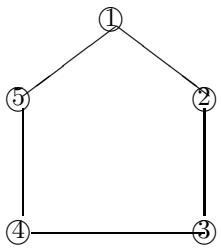


Figure 3.1: The interference graph of the program in Example 2

$v_3 := 4$

$v_2 := v_2 + v_3$

$v_3 := 5$

$v_4 := 6$

$v_3 := v_3 + v_4$

$v_4 := 7$

$v_5 := 8$

$v_4 := v_4 + v_5$

$v_5 := 9$

$v_1 := 10$

$v_5 := v_5 + v_1$

Definition 4 *(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.*

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

Definition 5 *(Cyclic mod n ordering) We use "$<_i$" to denote the cyclic mod n ordering relation for a fixed i, such that $i <_i (i+1) <_i \ldots <_i n <_i 1 <_i \ldots <_i i-1$. Where there is no confusion over the value of i, we omit the subscript in $<_i$.*

Definition 6 *(Potential watermark vertices [113, 114]) 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 $\forall j : i_1 <_i j <_i i_2$, $(v_i, v_j) \in E$.*

Definition 7 *(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 $\mathrm{PW}(v_i, G) =$ TRUE. Otherwise $\mathrm{PW}(v_i, G) =$ FALSE. When $\mathrm{PW}(v_i, G) =$ TRUE, the potential watermark vertices with respect to v_i are denoted as $\mathrm{PWV}(v_i, G, 1) = v_{i_1}$ and $\mathrm{PWV}(v_i, G, 2) = v_{i_2}$. Otherwise we say the values of $\mathrm{PWV}(v_i, G, 1)$ and $\mathrm{PWV}(v_i, G, 2)$ are undefined.*

Now we present the QP algorithm in [89].

Example 3 *(The QP embedding algorithm) If the interference graph of a program to be watermarked is G and the watermark to be inserted is W, then the QP algorithms in Fig. 3.2*

Example 4 *(The time complexity of the QP algorithm in Fig.3.2) This is an algorithm with linear complexity.*

To illustrate the QP algorithm, we present an example to show how to insert one bit by this algorithm.

Example 5 *After embedding a bit 0 into the interference graph of the program in Example 2 by the algorithm in Fig.3.2, we obtain the watermarked interference graph in Fig.3.3:*

The following is the corresponding program.

$v_1 := 1$

$v_2 := 2$

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 **then** // not all bits of W can be inserted in G

return G

for each i from 1 to n **do**

if j > m **then** // all bits of W already inserted in G

return G'

if $\mathrm{PW}(v_i, G')$ **then**

$G' := G' + (v_i, \mathrm{PWV}(v_i, G', w_j + 1))$

j++

if m ≥ j **then** // not all bits of W inserted in G

return G

return G'

Figure 3.2: A clarified version of the QP algorithm [113, 114]

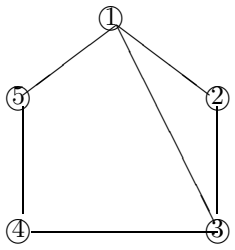


Figure 3.3: The result interference graph by inserting one bit "0"

$v_1 := v_1 + v_2$

$v_3 := 0$ *//Added statement*

$v_1 := v_1 + v_3$ *//Added statement*

$v_2 := 3$

$v_3 := 4$

$v_2 := v_2 + v_3$

$v_3 := 5$

$v_4 := 6$

$v_3 := v_3 + v_4$

$v_4 := 7$

$v_5 := 8$

$v_4 := v_4 + v_5$

$v_5 := 9$

$v_1 := 10$

$v_5 := v_5 + v_1$

Definition 8 *(Normal embedding) If $A(P, \epsilon) = P$, the embedder A is called normal.*

Example 6 *The QP embedding algorithm in Fig. 3.2 and the trivial embedder are normal.*

Definition 9 *(Set of candidate watermarks) A $W \in \mathbf{W}$ 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 A can actually insert into P. Embedding other watermarks into P will not change the original program.

Example 7 *(Set of candidate watermarks) Let A be the QP algorithm in Fig. 3.2 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 the original program and the watermarked programs are in Fig. 3.4. We can see from Fig. 3.4 that the interference graph for the watermarked program after inserting a watermark 010 is the same as that after inserting a watermark 111. For this reason a watermark embedded by the QP algorithm can not be extracted reliably.*

Example 8 *(Set of candidate watermarks of the trivial embedder)* $\forall P \in \mathbf{P}$,

$$candidate(P, Triv) = \phi.$$

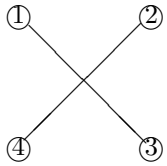
The trivial embedder is the only embedder A such that $\forall P \in \mathbf{P}$, $candidate(P, A) = \phi$.

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

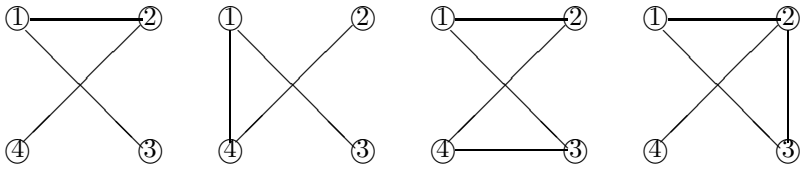
Le and Desmedt [68] developed a destroy algorithm to attack the QP embedding algorithm. The main result, Theorem 2 in [68], about this destroy algorithm is as follows:

Let C'' be the output of the destroy algorithm proposed by Le and Desmedt in [68] on input (G, C'), where C' is the output of the QP embedding algorithm [113, 114] on input graph G. Then the verification algorithm [68] will always output yes on input (G, C'', M) for arbitrary signature M.

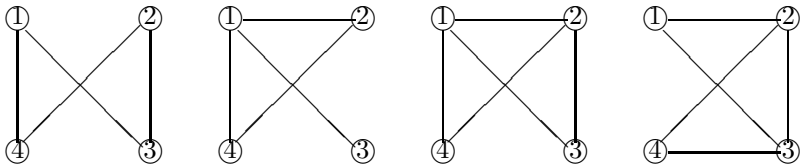
It is necessary to solve the contradiction between the arbitrary M in the above theorem and that for a graph with n vertices, the QP algorithm can insert at most only n bits of a message into such a graph. The concept of "set of candidate watermarks"



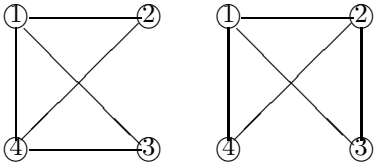
The original interference graph.



0 inserted. 1 inserted. 00 inserted. 01 inserted.



10 inserted. 11 inserted. 010 inserted. 011 inserted.



110 inserted. 111 inserted.

Figure 3.4: The interference graphs of the original and the watermarked programs

is one of a set that includes all watermarks which are actually 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 think we embed U instead W into this program.

Definition 10 *(Finite embedding) An embedder $A : \mathbf{P} \times \mathbf{W} \rightarrow \mathbf{P}$ 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 9 *The QP algorithm in Fig. 3.2 and the trivial embedder are finite.*

Definition 11 *(Representative sets) Let A be an embedder. For a program P, $A(P, W) = A(P, W')$, $W, W' \in \mathbf{W}$ is an equivalence relation in the set of candidate watermarks of P and A. Every equivalence 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 the 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 12 *(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 quality of an embedder A. The smaller it is, the better the A is.

Example 10 *(Representative sets and degree) A and P are as in Example 3. The representative sets of A and P are $\{0\}$, $\{1\}$, $\{00\}$, $\{01\}$, $\{10\}$, $\{11\}$, $\{011\}$, $\{110\}$, $\{010,111\}$. Thus, $repdegree(P, A) = 2$.*

3.3 Extracting

Definition 13 *(Extracting) Let A be an embedder, a function $X : \mathbf{P} \times \mathbf{P} \to \mathbf{W}$ is called an extracting algorithm corresponding to the embedder A if X has the following property:*

$\forall P, P' \in \mathbf{P}$, if $W \in candidate(P, A)$ and $P' = A(P, W)$, $X(P', P) = W$. Otherwise, $X(P', P) = \epsilon$.

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. From the definition of the extracting, 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 14 *(Extractable) 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.*

It is easy to prove the first part of this theorem from the definitions. Therefore we prove only the second part of this theorem.

Define an extracting algorithm corresponding to the embedding algorithm A as follows:

$\forall P', P \in \mathbf{P}$, if there is a $w \in candidate(P, A)$ such that $P' = A(P, W)$, $X(P', P) = W$.

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

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

Example 11 *(The QP algorithm in Fig. 3.2 is not extractable) From Example 10 and Theorem 1, The QP algorithm is not extractable.*

Example 12 *(Two extractable improvements on the QP algorithm) Because the QP algorithm is not extractable, we develop two improvements on the QP algorithm, the QPI algorithm and the QPII algorithm. They are extractable. The reason for the QP algorithm unextractability is the modulo n in the definition of the potential watermark vertices. We avoid it in two ways. First, as used in the QPI algorithm, we add two vertices to the original graph. Second, as used in the QPII algorithm, we do not use the modulo n; i.e., define the potential watermark vertices as follows:*

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 $i < i_1 < i_2 \leq n$; $(v_i, v_{i_1}) \notin E$; $(v_i, v_{i_2}) \notin E$; $\forall j : i < j < i_1$, $(v_i, v_j) \in E$; and $\forall j : i_1 < j < i_2$, $(v_i, v_j) \in E$.

The QPI embedding algorithm is in Fig. 3.5 and its corresponding extracting algorithm is in Fig. 3.6.

The QPII embedding algorithm is in Fig. 3.7 and its corresponding extracting algorithm is in Fig. 3.8.

Example 13 *(The time complexity of the QPI algorithm in Fig. 3.5) The time complexity of this algorithm is $O(n)$, where n is the number of edges of the interference graph of the program to be watermarked. In other words, the QPI algorithm is of linear complexity.*

Input: an original graph $G(V, E)$ with $n = |V|$

a message to be embedded into the $G(V, E)$: $W = w_1 w_2 \dots w_m$

Output: A watermarked graph G'.

Algorithm:

$G' := G$;

add two vertices v_{n+1}, v_{n+2} to V'

$j := 1$;

if m > n **then** // not all bits of W can be inserted in G

  return G

for each i from 1 to n **do**

  if j > m **then** // all bits of W already inserted in G

    exit

  if $\mathrm{PW}(v_i, G')$ **then**

    $G' := G' + (v_i, \mathrm{PWV}(v_i, G', w_j + 1))$

    j++

if m $\geq$ j **then** // not all bits of W inserted in G

  return G

return G'

Figure 3.5: The QPI Embedding Algorithm

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:
if G is not a subgraph of G' **then**
  return ϵ
add two vertices v_{n+1}, v_{n+2} to V
$j := 0$
for each i from 1 to n **do**
  if $\mathrm{PW}(v_i, G)$ **then**
    j++
    if $(v_i, v_{i_1}) \in E'$ **then**
      $w_j := 0$
      $G := G + (v_i, \mathrm{PWV}(v_i, G, w_j + 1))$
    else if $(v_i, v_{i_2}) \in E'$ **then**
      $w_j := 1$
      $G := G + (v_i, \mathrm{PWV}(v_i, G, w_j + 1))$
    else
      exit
if j=0 **then**
  return ϵ
if $|E'| \neq |E| + j$ **then**
  return ϵ
return the message $W = w_1 w_2 \ldots w_j$

Figure 3.6: The QPI Extraction Algorithm

Input: an original graph $G(V, E)$ with $n = |V|$

a message to be embedded into the $G(V, E)$: $W = w_1 w_2 \dots w_m$

Output: A watermarked graph G'.

Algorithm:

$G' := G$;

$j := 1$

if m $>$ n **then** // not all bits of W can be inserted in G

  return G

for each i from 1 to n **do**

  if j $>$ m **then** // all bits of W already inserted in G

    exit

  if PW(v_i, G') **then**

    $G' := G' + (v_i, \text{PWV}(v_i, G', w_j + 1))$

    j++

if m $\geq$ j **then** // not all bits of W inserted in G

  return G

return G'

Figure 3.7: The QPII Embedding Algorithm

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:

if G is not a subgraph of G' **then**

  return ϵ

$j := 0$

for each i from 1 to n **do**

  if $\mathrm{PW}(v_i, G)$ **then**

    j++

    if $(v_i, v_{i_1}) \in E'$ **then**

      $w_j := 0$

      $G := G + (v_i, \mathrm{PWV}(v_i, G, w_j + 1))$

    else if $(v_i, v_{i_2}) \in E'$ **then**

      $w_j := 1$

      $G := G + (v_i, \mathrm{PWV}(v_i, G, w_j + 1))$

    else

      exit

if j=0 **then**

  return ϵ

if $|E'| \neq |E| + j$ **then**

  return ϵ

return the message $W = w_1 w_2 \ldots w_j$

Figure 3.8: The QPII Extraction Algorithm

Example 14 *(The time complexity of the QPII algorithm in Fig. 3.7) Like the QPI algorithm, the time complexity of the QPII algorithm is of linear complexity.*

Definition 15 *(Blind and informed extracting) For an embedding algorithm* $A : \mathbf{P} \times \mathbf{W} \rightarrow \mathbf{P}$*, if there exists a function* $Y : \mathbf{P} \rightarrow \mathbf{W}$ *having the following properties:* $\forall P' \in \mathbf{P}$*,*

if $P' = A(P, W)$ *for a* $P \in \mathbf{P}$ *and a* $W \in candidate(P, A)$*,* $Y(P') = W$*.*

Otherwise, $Y(P') = \epsilon$*.*

then we say that watermarks embedded in programs of $\mathbf{P}$ *using an embedding algorithm* $A : \mathbf{P} \times \mathbf{W} \rightarrow \mathbf{P}$ *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 $X : \mathbf{P} \times \mathbf{P} \rightarrow \mathbf{W}$ *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 an* informed watermark extraction system*, when* X *is an informed extractor for* A*.*

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

Example 15 *(A not blindly extractable embedder) The QPI embedding algorithm in Fig. 3.5 is not blindly extractable.*

Example 16 *(A blindly extractable embedding algorithm) Define an embedder* A *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 a blind extracting algorithm X corresponding to A as follows:

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

3.4 Representative Extracting

Definition 16 *(Representative extracting) Let A be an embedder; a function $X : \mathbf{P} \times \mathbf{P} \rightarrow \mathbf{W}$ is called a representative extracting algorithm corresponding to the embedder A if it has the following property:*

$\forall W \in \mathbf{W}$ and $\forall P', P \in \mathbf{P}$, if $W \in candidate(P, A)$ and $P' = A(P, W)$, $X(A(P, X(P', P)), P) = X(P', P)$; otherwise, $X(P', P) = \epsilon$

The background for the representative extracting algorithms is as follows. For an embedding algorithm A, for example, we may get a same watermarked program P' after inserting any watermark in $\{101, 1110, 0010\}$ by A into a 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.

Property 3 *If X is an extracting algorithm corresponding to an embedder A, then X is also a representative extracting algorithm corresponding to A.*

Property 4 *(Characteristic of representative extracting algorithms) 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 2 *For every embedder A, there exists a representative extracting algorithm corresponding to A.*

$\forall W \in \mathbf{W}$ and $\forall P', P \in \mathbf{P}$, define a function $X : \mathbf{P} \times \mathbf{P} \rightarrow \mathbf{W}$ as follows.

Suppose $R_1, R_2, \ldots, R_i, \ldots$ are all representative sets of A and P. Choose one watermark W_1 from R_1, one watermark W_2 from R_2, ..., one watermark W_i from R_i, ... If $W \in candidate(P, A)$ and $P' = A(P, W)$, then there exists an i such that $W \in R_i$, so we define $X(P', P) = W_i$. Otherwise, $X(P', P) = \epsilon$.

It is easy to see such an X is a representative extracting algorithm corresponding to A.

From Theorem 2, we have the following concept.

Definition 17 *(Proper Representative Extractable) Let A be an embedder. It is called proper representative extractable if it is not extractable.*

Theorem 3 *An embedder A is proper representative extractable if and only if, for some program P, $repdegree(P, A) > 1$.*

Theorem 4 *An embedder A is proper representative extractable if and only if there are more than one representative extracting algorithms corresponding to A.*

Example 17 *(A QP representative extracting algorithm) The algorithm in Fig. 3.9 is a representative extracting algorithm corresponding to the QP embedding algorithm in Fig. 3.2. The extracting algorithm outlined in [113] is really not an extracting algorithm but a representative extracting algorithm. For the same program in Example 3, 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. As an example, another representative extracting algorithm for the QP algorithm is in Fig. 3.10. For a same program in

Example 3, 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.

Example 18 *(A proper representative extractable embedder) The QP algorithm in Fig. 3.2 is proper representative extractable. Example 11 shows the QP algorithm is not extractable.*

Definition 18 *(Informed and blind representative extracting) Let A be an embedder; a function $Y : \mathbf{P} \to \mathbf{W}$, satisfying,*

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

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 19 *(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 is blindly representative extractable. The combination (A, X) is called a blind representative watermark extraction system.*

Theorem 5 *If an embedding algorithm is blindly extractable, then it is also blindly representative extractable.*

Theorem 6 *If an embedding algorithm is proper representative extractable, then the representative degree of some program and A is greater than 1.*

Input: the original graph $G(V, E)$ with $n = |V|$ and the watermarked graph $G(V', E')$

Output: the message W embedded in the watermarked graph $G'(V', E')$

Algorithm:

$G'' := G$

if G is not a subgraph of G' **then return** "no watermark"

$j := 0$

for each i from 1 to n **do**

  if $\mathrm{PW}(v_i, G)$ **then**

    j++

    if $(v_i, v_{i_1}) \in E'$ **then**

      $w_j := 0$

      $G := G + (v_i, \mathrm{PWV}(v_i, G, w_j + 1))$

    else if $(v_i, v_{i_2}) \in E'$ **then**

      $w_j := 1$

      $G := G + (v_i, \mathrm{PWV}(v_i, G, w_j + 1))$

    else exit

if $|E'| = |E| + j$ **then return** the message $W = w_1 w_2 \ldots w_j$

$G := G''$

$j := 0$

for each i from 1 to n **do**

  if $\mathrm{PW}(v_i, G)$ **then**

    j++

    if $(v_i, v_{i_2}) \in E'$ **then**

      $w_j := 1$

      $G := G + (v_i, \mathrm{PWV}(v_i, G, w_i + 1))$

    else if $(v_i, v_{i_1}) \in E'$ **then**

      $w_j := 0$

      $G := G + (v_i, \mathrm{PWV}(v_i, G, w_i + 1))$

    else exit

if $|E'| = |E| + j$ **then**

  return the message $W = w_1 w_2 \ldots w_j$

return the message $W = \epsilon$

Figure 3.9: A QP representative extracting algorithm: '0' priority

Input: the original graph $G(V, E)$ with $n = |V|$ and the watermarked graph $G(V', E')$

Output: the message W embedded in the watermarked graph $G'(V', E')$

Algorithm:

if G is not a subgraph of G' **then return** "no watermark"

$j := 0$

for each i from 1 to n **do**

  if $\mathrm{PW}(v_i, G)$ **then**

    j++

    if $(v_i, v_{i_2}) \in E'$ **then**

      $w_j := 1$

      $G := G + (v_i, \mathrm{PWV}(v_i, G, w_j + 1))$

    else if $(v_i, v_{i_1}) \in E'$ **then**

      $w_j := 0$

      $G := G + (v_i, \mathrm{PWV}(v_i, G, w_j + 1))$

    else exit

if $|E'| = |E| + j$ **then return** the message $W = w_1 w_2 \dots w_j$

$G := G''$

$j := 0$

for each i from 1 to n **do**

  if $\mathrm{PW}(v_i, G)$ **then**

    j++

    if $(v_i, v_{i_1}) \in E'$ **then**

      $w_j := 0$

      $G := G + (v_i, \mathrm{PWV}(v_i, G, w_i + 1))$

    else if $(v_i, v_{i_1}) \in E'$ **then**

      $w_j := 1$

      $G := G + (v_i, \mathrm{PWV}(v_i, G, w_i + 1))$

    else exit

if $|E'| = |E| + j$ **then return** the message $W = w_1 w_2 \dots w_j$

return the message $W = \epsilon$

Figure 3.10: Another QP representative extracting algorithm: '1' priority

Example 19 *(A not blindly representative extractable embedder) The QP algorithm in Fig. 3.2 is not blindly representative extractable.*

Example 20 *(A blindly representative extractable embedder) Define an embedder A 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 $P' \in \mathbf{P}$, if P' has at least one variable declaration, $Y(P') = 101$. Otherwise, $Y(P') = \epsilon$.

3.5 A Software Watermark Embedding and Extracting System

We have developed a model to illustrate how our embedding and extraction algorithms could be used in a complete system for software watermarking. The process for the embedding subsystem goes:

For $\forall P \in \mathbf{P}$ and $\forall W \in \mathbf{W}$,

Step 1: Construct the interference graph G of P.

Step 2: Embed the watermark W into the graph G by the QPI embedding algorithm or the QPII embedding algorithm to get the watermarked graph G'.

Step 3: Establish interference relationships of some variable pairs in P so that the interference graph of the new program is G' as did in paper [89].

The process for the extracting subsystem is:

For $\forall P, P' \in \mathbf{P}$,

Step 1: Construct the interference graphs G, G' of P, P', respectively.

Step 2: Extract the watermark W from the graphs G and G' by the QPI extracting algorithm or the QPII extracting algorithm.

3.6 Conclusions

Algorithmic design, even with an adequate formal statement of the problem to be solved, is as much art as it is 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 were successful 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. 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. Some other concepts are also introduced in this chapter.

We have presented a model for a complete watermarking system which is based on the concepts and algorithms established in this chapter. We will implement and analyse it in our future work. In the next chapter, we will study the recognition of

watermarks in software.

Though we discussed the time complexities of the QP algorithm, the QPI algorithm, and the QPII algorithm, the complexity of software watermarking algorithms is still an important research issue. We leave it for our future work. All embedding and extraction algorithms discussed in this chapter are blind algorithms, i.e., in addition to the watermarked program, we need the original program for extraction. But we do not need any key for extraction. As for the security of the watermarking embedding algorithms, Barak et al. [10] have already proved that it is impossible to construct a one-way software watermarking transformation in their model. But we still can develop software watermarking algorithms for special purposes.

Chapter 4

Recognition

When we are presented software, how can we say whether it contains a watermark or not? When we are not sure, what is the probability of it containing a certain kind of watermark? We formalize the concept of recognition in software watermarking to help us find answers to these questions. In the previous chapter, we were interested in extracting every bit of a watermark embedded in a program, but, in some situations, we are interested in knowing if a watermark exists in a program or not. We have already published this chapter in paper [171].

This chapter is organized as follows. In Section 4.1, we define recognition, partial recognition, blind recognition, and blind partial recognition of software watermarks. Here we also develop several recognition and partial recognition algorithms for a software watermarking algorithm, called the QP algorithm. In Section 4.2, we describe a model of a software watermark embedding and recognition system based on the concepts and algorithms developed in this chapter. Section 4.3 has the summary of this chapter.

4.1 Recognition

In some situations, we may not have to extract all bits of a watermark inserted in software; we just want to see whether software contains a watermark inserted by an embedding algorithm. In some other situations, the software watermark embedding algorithm may be in essence not extractable, or, even when the embedding algorithm is extractable, the extracting algorithm may be not so efficient for a specific application. In this section, we discuss the problem of determining the existence of a watermark embedded in a software program.

4.1.1 Recognitions and Partial Recognitions

Definition 20 *(Partial recognition) For an embedding algorithm* $A : \mathbf{P} \times \mathbf{W} \rightarrow \mathbf{P}$*, if a function* $R : \mathbf{P} \times \mathbf{P} \rightarrow \{\text{TRUE}, \text{FALSE}\}$*, satisfies that* $\forall P, P' \in \mathbf{P}$*, if there is a* $W \in candidate(A, P)$ *such that* $P' = A(P, W)$ *then* $R(P', P) = \text{TRUE}$*, we call* R *a* positive-partial *recognition algorithm for the embedding algorithm* A*.*

For an embedding algorithm $A : \mathbf{P} \times \mathbf{W} \rightarrow \mathbf{P}$*, if a function* $R : \mathbf{P} \times \mathbf{P} \rightarrow \{\text{TRUE}, \text{FALSE}\}$*, satisfies* $\forall P, P' \in \mathbf{P}$*,* $R(P', P) = \text{TRUE} \implies P' = A(P, W)$ *for some* $W \in candidate(A, P)$*, we call* R *a* negative-partial *recognition algorithm for the embedding algorithm* A*.*

Definition 21 *(Recognition) For an embedding algorithm* $A : \mathbf{P} \times \mathbf{W} \rightarrow \mathbf{P}$*, if a function* $R : \mathbf{P} \times \mathbf{P} \times \mathbf{W} \rightarrow \{\text{TRUE}, \text{FALSE}\}$ *is both a positive-partial recognition and a negative-partial recognition, i.e.,* R *satisfies* $\forall P, P' \in \mathbf{W}$*,* $R(P', P) = \text{TRUE} \iff P' = A(P, W)$ *for some* $W \in candidate(A, P)$*, we call* R *a* recognition algorithm *for the embedding algorithm* A*.*

We say that A *is recognizable if a recognition algorithm exists for* A*.*

Example 21 *(Trivial partial recognitions) The partial recognition concepts are very flexible. The following are some trivial partial recognitions. For an embedding algorithm* $A : \mathbf{P} \times \mathbf{W} \rightarrow \mathbf{P}$*, define a function* $S : \mathbf{P} \times \mathbf{P} \rightarrow \{\text{TRUE}, \text{FALSE}\}$*, as* $P', P \in \mathbf{P}$*,* $S(P', P) = \text{TRUE}$*. This is a positive-partial recognition corresponding to* A*. We call such a function a trivial positive-partial recognition corresponding to* A *and denote it as* $TrivPP(A)$*.*

For an embedding algorithm $A : \mathbf{P} \times \mathbf{W} \rightarrow \mathbf{P}$*, define a function* $S : \mathbf{P} \times \mathbf{P} \rightarrow \{\text{TRUE}, \text{FALSE}\}$ *as* $P', P \in \mathbf{P}$*,* $S(P', P) = \text{FALSE}$*. This is a negative-partial recognition corresponding to* A*. We call such a function a trivial negative-partial recognition and denote it as* $TrivNP(A)$*.*

Theorem 7 *For every embedder* A*, there exists one and only one recognition algorithm corresponding to* A*. We denote the unique recognition algorithm corresponding to* A *as* $Reg(A)$*.*

Proof. $\forall P, P' \in \mathbf{P}$, define $R(P', P)$ as follows.

$R(P', P) = \text{TRUE}$, if there is some $W \in candidate(A, P)$ such that $P' = A(P, W)$

$R(P', P) = \text{FALSE}$, otherwise

It is easy to see R is a recognition algorithm corresponding to A.

From Theorem 7 and Example 11, not all embedding algorithms are extractable, but, in a sense, every embedding algorithm is recognizable.

Property 5 *For every embedder* A*,* $Reg(A)$ *is both a positive-partial and a negative-partial recognition algorithms corresponding to* A*.*

Example 22 *(A positive-partial recognition for the QP algorithm) A positive-partial recognition for the QP algorithm is in Fig. 4.2. For the program with its interference graph as in Example 7, the programs recognized by this recognition algorithm have interference graphs as in Fig. 4.1.*

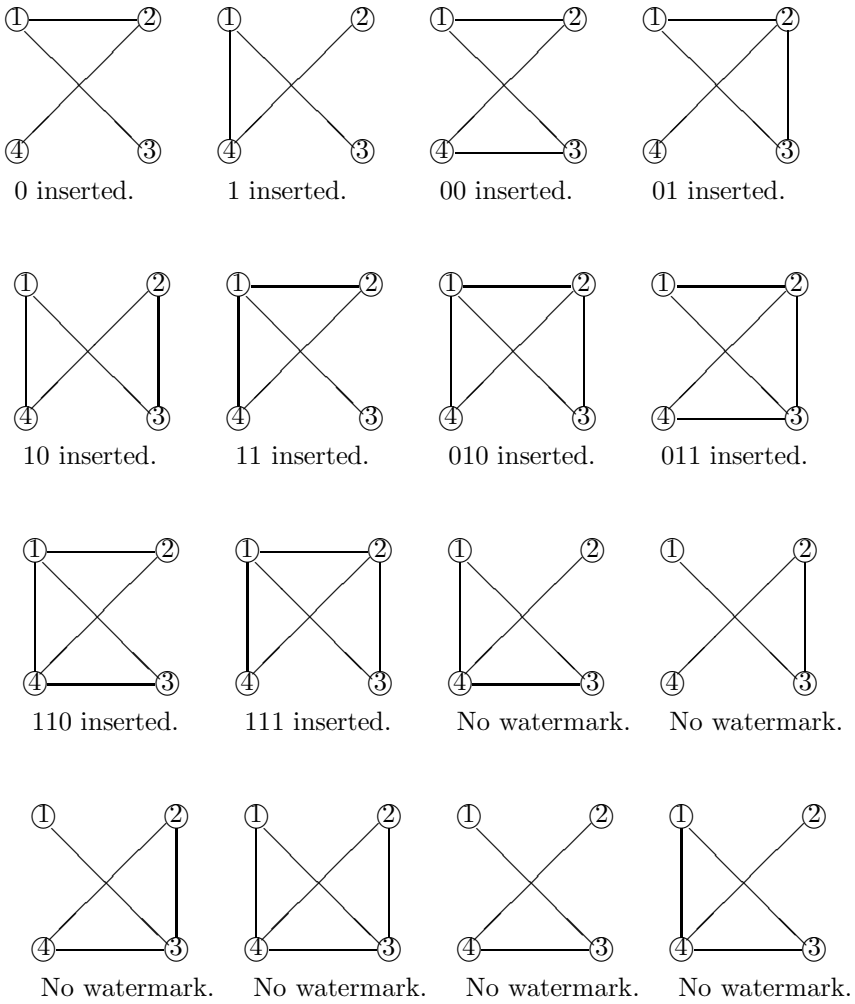
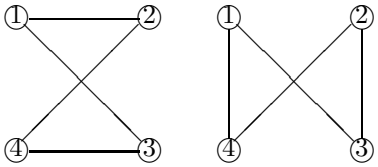


Figure 4.1: The interference graphs for watermarked programs recognized as TRUE

Input: an unwatermarked graph $G(V, E)$ with $n = |V|$
a watermarked graph G'
Output: is a message W embedded in G'?
Algorithm:
if G is not a subgraph of G' **then**
  return FALSE
j:=0
for each i from 1 to n **do**
  if find the nearest two vertices v_{i_1}, v_{i_2} not connected to v_i in G **then**
    j++
    if $(v_i, v_{i_2}) \in G'$ **then**
      connect v_i to v_{i_2} in G
    else if $(v_i, v_{i_1}) \in G'$ **then**
      connect v_i to v_{i_1} in G
    else // all bits extracted
      exit
if j=0 **then**
  return FALSE
return TRUE

Figure 4.2: A positive-partial recognition for the QP algorithm



00 inserted. 10 inserted.

Figure 4.3: The interference graphs for watermarked programs recognized as TRUE

Example 23 *(The time complexity of the positive-partial recognition for the QP algorithm) The recognition algorithm in Fig. 4.2 is of linear time complexity.*

Example 24 *(A negative-partial recognition for the QP algorithm) A negative-partial recognition for the QP algorithm is in Fig. 4.4. For the program with its interference graph as in Example 7, the programs recognized by this recognition algorithm have interference graphs as in Fig. 4.3.*

Example 25 *(A recognition for the QP algorithm) A recognition for the QP algorithm is in Fig. 4.5.*

Definition 22 *(Strength of partial recognitions) Let* $PP1$ *and* $PP2$ *be two positive-partial recognitions corresponding to an embedding algorithm* A*; if* $\forall P, P' \in \mathbf{P}$, $PP2(P', P) = \text{TRUE} \implies PP1(P', P) = \text{TRUE}$*, we say* $PP2$ *is stronger than* $PP1$*.*

Let $NP1$ *and* $NP2$ *be two negative-partial recognitions corresponding to an embedding algorithm* A*; if* $\forall P, P' \in \mathbf{P}$, $NP1(P', P) = \text{TRUE} \implies NP2(P', P) = \text{TRUE}$*, we say* $NP2$ *is stronger than* $NP1$*.*

Property 6 *For every embedder* A*,* $TrivPP(A)$ *is the weakest positive-partial recognition and* $Reg(A)$ *is the strongest positive-partial recognition for* A*;* $TrivNP(A)$ *is the weakest negative-partial recognition and* $Reg(A)$ *is the strongest negative-partial recognition for* A*.*

Input: an unwatermarked graph $G(V, E)$ with $n = |V|$
a watermarked graph G'
Output: is a message W embedded in G'?
Algorithm:
if G is not a subgraph of G' **then**
return FALSE
j:=0
for each i from 1 to n **do**
if find the nearest two vertices v_{i_1}, v_{i_2} not connected to v_i in G **then**
j++
if $(v_i, v_{i_1}) \in G'$ **then**
connect v_i to v_{i_1} in G
else if $(v_i, v_{i_2}) \in G'$ **then**
connect v_i to v_{i_2} in G
else
return FALSE
if j=0 **then**
return FALSE
return TRUE

Figure 4.4: A negative-partial recognition for the QP algorithm

Input: an unwatermarked graph $G(V, E)$ with $n = |V|$

a watermarked graph G'

Output: is a message W embedded in G'

Algorithm:

if G is not a subgraph of G' **then**

  return FALSE

j:=0

for each i from 1 to n **do**

  if find the nearest two vertices v_{i_1}, v_{i_2} not connected to v_i in G **then**

    j++

    if $(v_i, v_{i_2}) \in G'$ **then**

      connect v_i to v_{i_2} in G

    else if $(v_i, v_{i_1}) \in G'$ **then**

      connect v_i to v_{i_1} in G

    else // all bits extracted

      exit

if j=0 **then**

  return FALSE

if $|E'| \neq |E| + j$ **then**

  return FALSE

return TRUE

Figure 4.5: A recognition for the QP algorithm

4.1.2 Blind Recognitions

Definition 23 *(Blind and informed recognition) For an embedding algorithm* $A : \mathbf{P} \times \mathbf{W} \rightarrow \mathbf{P}$ *and a function* $S : \mathbf{P} \rightarrow \{\text{TRUE}, \text{FALSE}\}$ *we make the following definitions.*

If S *satisfies the following property:*

$\forall P' \in \mathbf{P}$*, if there is a* $P \in \mathbf{P}$ *and a* $W \in candidate(A, P)$ *such that* $P' = A(P, W)$*, then* $S(P') = \text{TRUE}$*.*

then we call S *a* blind positive-partial *recognition algorithm for the embedding algorithm* A*.*

If S *satisfies that*

$\forall P' \in \mathbf{P}$, $S(P') = \text{TRUE} \implies$ *there is a* $P \in \mathbf{P}$ *and a* $W \in \mathbf{W}$ *such that* $P' = A(P, W)$*'.*

then we call S *a* blind negative-partial *recognition algorithm for the embedding algorithm* A*.*

If S *satisfies that:*

$\forall P' \in \mathbf{P}$, $S(P') = \text{TRUE} \iff$ *there is a* $P \in \mathbf{P}$ *and a* $W \in candidate(A, P)$*, such that* $P' = A(P, W)$*.*

then we call S *a blind recognition algorithm for the embedding algorithm* A*.*

We say that A *is blind recognizable if a blind recognition algorithm exists for* A*.*

The combination (A, S) *is called a* blind watermark recognition system*, when* S *is a blind recognizer for* A*.*

Blind recognition is of more practical interest than informed recognition, for the same reasons that blind extraction is more useful than informed extraction.

Example 26 *(Trivial blind partial recognitions) The blind partial recognition concepts are also very flexible. The following are some trivial blind partial recogni-*

tions. For an embedding algorithm $A : \mathbf{P} \times \mathbf{W} \rightarrow \mathbf{P}$*, define a function* $S : \mathbf{P} \rightarrow \{\text{TRUE}, \text{FALSE}\}$*, as*

$\forall W \in \mathbf{W}$, $S(P') = \text{TRUE}$. *This is a blind positive-partial recognition corresponding to* A*. We call such a function a trivial blind positive-partial recognition and denote it as* $TrivBPP(A)$*.*

For an embedding algorithm $A : \mathbf{P} \times \mathbf{W} \rightarrow \mathbf{P}$*, define a function* $S : \mathbf{P} \rightarrow \{\text{TRUE}, \text{FALSE}\}$ *as* $\forall P' \in \mathbf{P}$, $S(P') = \text{FALSE}$. *This is a blind negative-partial recognition corresponding to* A*. We call such a function a trivial blind negative-partial recognition and denote it as* $TrivBNP(A)$*.*

Example 27 *(A blind recognition) Define an embedder* A *as follows.*
For any program P*, if* $W = 101$ *or if* $W = 110$*,* $A(P, W)$ *is* P *plus an extra constant declaration. Otherwise,* $A(P, W) = P$*. A blind recognition algorithm* S *corresponding to* A *is defined as follows.*
For any $P' \in \mathbf{P}$*, if* P' *has at least one constant declaration,* $S(P') = \text{TRUE}$*. Otherwise,* $Y(P') = \text{FALSE}$*.*

Theorem 8 *For every embedder* A*, there exists one and only one blind recognition algorithm corresponding to* A*. We denote the unique blind recognition algorithm corresponding to* A *as* $BReg(A)$*.*

Proof. $\forall P, P' \in \mathbf{W}$, define $R(P', P)$ as follows:

if there is a $P \in \mathbf{P}$ and a $W \in candidate(A, P)$ such that $P' = A(P, W)$, $R(P', P) = \text{TRUE}$,

Otherwise, $R(P', P) = \text{FALSE}$.

It is easy to see R is a blind recognition algorithm corresponding to A.

Property 7 *For every embedder* A*,* $BReg(A)$ *is both a blind positive-partial and a negative-partial recognition algorithms corresponding to* A*.*

Definition 24 *(Strength of blind partial recognitions) Let* $BPP1$ *and* $BPP2$ *be two blind positive-partial recognitions corresponding to an embedding algorithm* A*; if* $\forall W \in \mathbf{W}$ *and* $\forall P' \in \mathbf{P}$, $BPP2(P') = \text{TRUE} \implies BPP1(P') = \text{TRUE}$*, we say* $BPP2$ *is stronger than* $BPP1$*.*

Let $BNP1$ *and* $BNP2$ *be two blind negative-partial recognitions corresponding to an embedding algorithm* A*; if* $\forall W \in \mathbf{W}$ *and* $\forall P' \in \mathbf{P}$, $BNP1(P') = \text{TRUE} \implies BNP2(P') = \text{TRUE}$*, we say* $BNP2$ *is stronger than* $BNP1$*.*

Property 8 *For every embedder* A*,* $TrivBPP(A)$ *is the weakest blind positive-partial recognition and* $BReg(A)$ *is the strongest blind positive-partial recognition;* $TrivBNP(A)$ *is the weakest blind negative-partial recognition and* $BReg(A)$ *is the strongest blind negative-partial recognition.*

4.2 A Software Watermark Embedding and Recognition System

We have developed a model to illustrate how our embedding and recognition algorithms could be used in a complete system for software watermarking.

The process of the embedding subsystem goes:

For $\forall P \in \mathbf{P}$ and $\forall W \in \mathbf{W}$,

Step 1: Construct the interference graph G of P.

Step 2: Embed the watermark W into the graph G by the QP embedding algorithm and we have the watermarked graph G'.

Step 3: Establish interference relationships of some variable pairs in P so that the interference graph of the new program is G'.

The process of the recognition subsystem is as follows:

For $\forall P, P' \in \mathbf{P}$,

Step 1: Construct the interference graphs G, G' of P, P', respectively.

Step 2: Recognize the watermark W from the graphs G and G' by one of the QP recognizing algorithm.

4.3 Conclusions

We already discussed a similar issue, extraction, in the last chapter, but recognition is still an interesting research problem. For the QP extraction and recognition algorithms discussed in this thesis, they are different in the following aspect; if a program has a watermark, an extraction algorithm stops only after all bits of this watermark have been checked, while a recognition algorithm stops as soon as the first bit of this watermark has been checked.

As we can see from the above definitions, recognition is not a trivial concept in software watermarking. How to construct a good recognition algorithm for a specific situation and purpose still deserves further research. Our major findings are summarized very briefly below.

For any software watermark embedding algorithm, there is one and only one recognition algorithm corresponding to it. This recognition algorithm is also the strongest positive-partial and the strongest negative-partial recognition corresponding to that embedding algorithm. There are also one weakest positive-partial recognition and one negative-partial recognition corresponding to that embedding algorithm; they are all trivial partial recognitions. Similar results hold for the blind recognition and partial recognitions.

Recognition is more flexible than extraction, but it is not at all obvious how to develop a good recognition algorithm for a specific situation. We model for a software watermark embedding and recognition system through register allocation, based on the concepts and algorithms established in this chapter. We will implement and

analyse it in our future work.

Though the QP recognition algorithms in this chapter have the same asymptotic time complexity as the QP extraction algorithm in previous chapter, generally, recognition will be somewhat faster than the corresponding extraction. Extraction focuses on the retrieval of the watermark that was embedded, while recognition is focused on the possibility that a watermark has been embedded. Furthermore, not all software embedding algorithms have corresponding extraction algorithms, but they certainly have corresponding recognition algorithms. In these situations, recognition algorithms are especially important.

This chapter concludes our discussion on software watermarking. In the next three chapter, we will study another similar issue in software security: software obfuscation. Software obfuscation can be used to protect watermark inserted in a program. The techniques in software obfuscation can also applied to software watermarking and vice verse.

Part II

Software Obfuscation

Chapter 5

Survey of Software Obfuscation

In the last three chapters, we studied one technique in software security: software watermarking. When a watermark is inserted into software, an attacker may use reverse engineering tools to locate the watermark embedded in software and try to delete this watermark. An attacker can analyse source code of software to find watermarks in software. He can also analyse execution trace of software to find an execution history of the software, such as function entries and exits, branch points and decisions. Such information can be used to understand the software and find watermarks. Thus, after inserting a watermark into software, our concern is about how to make it hard for an attacker to find and destroy a watermark in software. In order to make software watermark robust, we use several methods to keep it hard to find. One of these methods is software obfuscation. Software obfuscation is a way of protecting software from unauthorized modification. It translates a program into another one which is semantically equivalent but is hard for attackers to understand and analyze the obfuscated program.

In next three chapters, we will focus on software obfuscation. Firstly, an overview of this topic is presented. Then we detail a technique of obfuscating integer variables

and apply this method to array obfuscation.

5.1 Why Obfuscate Software?

Since Sun Microsystems designed Java in the mid-1990s, it has been widely used to deliver interactive web content on the Internet, such as video displays, animations, and interactive games. As the Internet is connected with various heterogeneous hardware with different architectures, Java is intended to be architecture-independent.

Traditionally, software developers first write source code for the software in some high level programming language, then compile the source code into native code which will execute only on a specific type of computer. With Java, the finished product of software, bytecode, is actually not the code in the architecture-specific machine language which can be understood by only a specific type of computers. Instead, bytecode is something that can be run on a "Java virtual machine" (JVM), a platform between the high-level language and the real computer. It is the JVM that makes the distributed executables of Java programs portable, not architecture-dependent.

Now that Java bytecode is architecture-independent, it contains much information of the source codes. This makes it easy to decompile Java bytecode into Java source code and extract important algorithms from it. This feature of Java bytecode helps attackers reverse-engineer Java bytecode and results in software piracy, unauthorized penetration and system modification.

After mobile code became popular in the Internet age, another type of software piracy occurred. As we know, mobile code migrates across a network from a remote source to a local system and is then run on that local system. The local system can be a personal computer, or a mobile phone, or an Internet appliance, so software developers and owners may encounter piracy from malicious hosts.

In order to protect software, especially that in forms such as Java bytecode and mobile code, several measures have been proposed. Among them software obfuscation seems a last approach. Software obfuscation transforms a program into another semantically equivalent one that is harder to understand and reverse engineer [33].

5.2 Definition of Software Obfuscation

Definition 25 *(Software obfuscation [33]) Let* $\mathbf{P}$ *be all programs. An algorithm* $O : \mathbf{P} \longrightarrow \mathbf{P}$ *is called a software obfuscation transformation if the following three conditions hold:*

- *For any given source program* $P \in \mathbf{P}$, $P' = O(P)$ *satisfies that if* P *fails to terminate with an error condition, then* P' *may or may not terminate. Otherwise,* P' *must terminate and produce the same output as* P.
- *The run time of* P' *should be at most polynomially larger than that of* P.
- *The time taken by an attacker to recover* P *from* P' *should be at least as large as the time to develop* P *from scratch.*

When $P' = O(P)$, P is called the unobfuscated program and P' is called the obfuscated program.

5.3 Taxonomy of Software Obfuscation

Four principal types of software obfuscations are design obfuscation, data obfuscation, control obfuscation, and layout obfuscation.

Design obfuscation [127] includes techniques such as class merging, class splitting and type hiding which will obfuscate the design intent of object-oriented software. Class merging obfuscation transforms a program into another one by merging two

or more classes in the program into a single class. Class splitting obfuscation splits a single class in a program into several classes. Type hiding obfuscation uses Java interfaces to obscure the design intent. Constants, variables and data structures are essential elements of a program. Data obfuscations [33] try to obfuscate data types and structures in a program. The common techniques for data obfuscations are split variable, merge variable, flatten array, fold array, promote scalar to object, convert static data to procedure, and change variable lifetime. With split variable, one variable is expressed in two or more variables. In merge variable, two or more variables are expressed in one variable. Split array expresses one array in two or more arrays. Merge array uses one array to represent two or more arrays. Flatten arrays uses a lower-dimensional array variable for a higher-dimensional array and fold array expresses a lower-dimensional array variable in a higher-dimensional array. The method of promote scalars to objects expresses one simple type of variable in an object with the same type. In contrast, converting static data to procedure replaces a static datum with a procedure which produces this datum. Lastly, changing variable lifetime converts a global variable to a local variable.

Information about control transitions in a program is central to exposing the locations and interpretations of sensitive states. To protect such information is an important issue in software security. A program can be divided into basic blocks. A basic block of a program is a sequence of codes without branches. In this way, the control information of a program can be represented as a Control Flow graph (CFG) which is a graph of basic blocks where the edges of the CFG is possible control flows between basic blocks. Obfuscation of the control flow and purpose of variables in a program can hide the intentions of a program. The followings are some techniques for control obfuscation [75]:

1. Reducible to non-reducible control flow graphs: Insert goto statement in Java bytecode.

2. Extended loop condition: Add opaque predicate in a loop to change control flow graph but to keep the semantics of the program.

3. Table interpretation: Choose a section of code in the unobfuscated program, then create a virtual process not existing in the host language and add a pseudo-code interpreter for such a virtual process into the obfuscated program.

4. Inline method: For every method call in a program, delete the method call and add a different copy of code for this method.

5. Outline statements: Convert a sequence of codes in a program into a method call.

6. Reorder block: Replace a static variable with a procedure.

7. Reorder loop: Convert a global variable to a local variable.

8. Intra-procedural transformations: Degeneration of control flow by changing static branches on dynamic and loose injection of data aliases.

10. Inter-procedural transformations: Convert a global variable to a local variable.

11. Creation of aliases to function pointers and data aliases.

The last type of software obfuscation is layout obfuscation [53]. It tries to obfuscate the lexical structure of the software by changing source code formatting, renaming variables, and removing debugging information.

5.4 Criteria of Software Obfuscation

When a software obfuscation approach is proposed, people need to evaluate it according to certain criteria. Different settings have different requirements for software obfuscation approaches. The following are the common criteria for software obfuscation [33].

Let $\mathbf{P}$ be all programs and $O : \mathbf{P} \longrightarrow \mathbf{P}$ a software obfuscation transformation. $P \in \mathbf{P}$ and $P' = O(P)$.

In the criterion of semantic-preservation, P' has the same observable behavior as P. With the obscurity criterion, to understand and reverse engineer P' is strictly more time-consuming than to understand and reverse engineer P. The criterion of resilience means it is either difficult to construct an automatic tool to undo the obfuscation transformation O, or executing such a tool is extremely time-consuming. In stealth, the statistical properties of P' are similar to those of P, and lastly, cost means that the Execution time/space penalty incurred by the obfuscation of P are small.

5.5 Status of Software Obfuscation

As pioneers in software obfuscation, Collberg, Thomborson et al. explored this research area in papers [28, 32, 33, 75]. These works included a detailed discussion on definitions, problems, techniques, and criterion in software obfuscation. The techniques in these papers mainly applied to object-oriented programs such as Java programs. Though these papers presented a comprehensive practical study for software obfuscation, they lacked theoretical foundation for this research. Based on that aliases in a program is an NP-hard, Wang et al. [138, 140] presented a software obfuscation method through global arrays and pointers. Their techniques applied to programs written in a programming language such as C which has pointers. Drape [43] proposed a series of techniques for data structure obfuscation based on established work on program refinement, abstract data-types and functional programming. The focus of his work was on the correctness of the proposed software obfuscation. Wroblewski [144] studied software obfuscation for machine code programs based on a heuristic approach. Sosonkin, Naumovich, and Memon [127] had a detailed discussion on design obfuscation.

In 2001, Barak et al. published the first formal study of software obfuscation based

on a cryptographic model called the virtual black box [10]. They presented several important impossibility results about software obfuscation. Goldwasser and Kalai proved the impossibility of such an obfuscation even with auxiliary input in Baraks cryptographic model [51]. Though Barak et al. proved the nonexistence of a universal obfuscator based on their model, there are still some positive results, and Barak et al.'s model does not apply to all settings software obfuscation practioners face. Lynn et al. presented a positive result for obfuscating point functions with a random oracle in paper [76]. Wee [141] provided a simple construction of efficient obfuscators for point functions for a slightly relaxed notion of obfuscation, for which obfuscating general circuits is nonetheless impossible. This construction relies on the existence of a very strong one-way permutation and yields the first non-trivial obfuscator under general assumptions in the standard model.

Other practical techniques to obfuscate software abound in [1, 4, 17, 19, 22, 45, 46, 48, 49, 53, 58, 61, 79, 78, 81, 95, 97, 98, 111, 112, 118, 121, 122, 134, 187]. As in a game of hide and seek, people also explore deobfuscation techniques [23, 63, 64, 136].

5.6 Conclusion

Software obfuscation is a way of protecting software piracy by making software programs hard for adversaries to understand. Though researchers have developed several techniques for software obfuscation, it is still a relatively new field and deserves further exploration. In this thesis, we focus on developing obfuscation techniques to hide important constants and arithmetic operations in programs and to obfuscate data structures in software.

Chapter 6

Homomorphic Functions

In software obfuscation, variable transformation is a major method to transform certain variables in a program into new ones so that it is hard for attackers to understand the true meaning of the variables [33]. Residue number coding [62] is an approach used in hardware design, high precision integer arithmetic, and cryptography. It is also used for software obfuscation [21, 97], but there is a big flaw in this application of residue number coding to data flow transformations. In this chapter, we propose the concepts of homomorphic obfuscations, a potential area for further exploration. Based on these concepts, we establish a sound grounding for residue number coding for software obfuscation and a precise relationship between the natural orders of residue numbers and homomorphic obfuscation. We also use this method to develop an algorithm for division by several constants. Some of the of results in this chapter have already been published [179].

This chapter is structured as follows. In Section 6.1 we describe basic concepts about residue numbers. In Sections 6.2 and 6.3 we establish a systematic theory for a solution to obfuscating addition, subtraction, and multiplication of residue numbers. Section 6.4 has an explanation for the difficulty of obfuscating the order operations

for integers through homomorphic obfuscations. The technique to obfuscate division of integers by a constant is presented in Section 6.5, while in Section 6.6 we propose a solution to the division of residue numbers which is easy to implement. This chapter concludes in section 6.7.

6.1 Basic Concepts about Residue Numbers

Let Z be the set of all integers, n a given positive integer. For any $x \in Z$, denote $[x]_n = \{y \in Z \mid y - x \text{ is divisible by } n\}$ and we call $[x]_n$ the residue class of x modulo n. We omit the subscript when there is no confusion. Let Z/nZ be the set of all these residue classes with respect to modulo n, where $Z/nZ = \{[0], [1], \ldots, [n-1]\}$.

For Z/nZ, we introduce a *natural order* relation $\leq$ among its members as $[0] \leq [1] \leq \ldots \leq [n-1]$

Z/nZ has three operations +, -, $\times$ defined as follows: for any two $[x], [y] \in Z/nZ$, $[x] + [y] = [x + y]$, $[x] - [y] = [x - y]$, $[x] \times [y] = [x \times y]$.

The product $Z/m_1Z \times Z/m_2Z \times \ldots \times Z/m_kZ$ also has three operations $+$, $-$, $\times$ defined as follows: for any two $([x_1]_{m_1}, \ldots, [x_k]_{m_k})$, $([y_1]_{m_1}, \ldots, [y_k]_{m_k}) \in Z/m_1Z \times \ldots \times Z/m_kZ$,

$$\begin{aligned}
([x_1]_{m_1}, \ldots, [x_k]_{m_k}) + ([y_1]_{m_1}, \ldots, [y_k]_{m_k}) &= ([x_1 + y_1]_{m_1}, \ldots, [x_k + y_k]_{m_k}) \\
([x_1]_{m_1}, \ldots, [x_k]_{m_k}) - ([y_1]_{m_1}, \ldots, [y_k]_{m_k}) &= ([x_1 - y_1]_{m_1}, \ldots, [x_k - y_k]_{m_k}) \\
([x_1]_{m_1}, \ldots, [x_k]_{m_k}) \times ([y_1]_{m_1}, \ldots, [y_k]_{m_k}) &= ([x_1 \times y_1]_{m_1}, \ldots, [x_k \times y_k]_{m_k})
\end{aligned}$$

In software obfuscation, sometimes we need to hide some constants. Generally, we will use coding methods to encode these constants and decode them. A simple example is shown in Fig. 6.2.

For the unobfuscated program in Fig. 6.1, an attacker might try to learn a valid license key "x". If this attacker puts a break point at the statement "if((c*x+d)%n!=e)",

```
void licenseCheck(int x){
   const int c = 5, d = 6, e = 7, n = 99;
   if((c*x+d)%n!=e)
       exit();
}
```

Figure 6.1: An unobfuscated function for license key checking

Obfuscated program

```
void obfLicenseCheck(int xe){
   const int ce = obf(5),de = obf(6), ee = obf(7);
   if(obfNeq(obfMod99(obfAdd(obfMulti(ce,xe),de),ee))
       exit();
}
```

Figure 6.2: Obfuscated version of Fig. 6.1

they can learn the license-checking equation and its constants. If they are able to solve this equation for x, they will be able to compute a valid license key. License keys should, of course, only be revealed to licensed users. So it is important to find ways to prevent such attacks on the key-checking function.

By obfuscating the constants and variables in Fig. 6.1, and by replacing its integer arithmetic operations with their equivalents, we could obtain the obfuscated program in Fig. 6.2. In this program, "obf(5)", "obf(6)", and "obf(7)" will be replaced by corresponding new concrete constants by compiler, so "obf" and all these original constants will disappear in this program. The attacker may still put a break point at the statement "if(obfNeq(obfMod99(obfAdd(obfMulti(ce,xe),de),ee))", but this statement and its constants are obfuscated. The attacker must work harder to compute the license key from this statement. At least, it requires more mathematical

```
void licenseCheck2(int x, int y)
   const int c = 5, d = 6, e = 7;
   if((c*x+d)%y!=e)
      exit();
}
```

Figure 6.3: Another program for license key checking. It takes two secret keys (x,y).

```
void obfLicenseCheck2(int xe, int ye){
   const int ce = obf(5),de = obf(6), ee = obf(7);
   int te = obfAdd(obfMulti(ce,xe),de));
   int t = deobf(te);
   int y = deobf(ye);
   int ze = obf(t% y);
   if(obfNeq(ze, ee))
      exit();
}
```

Figure 6.4: An example of improper data obfuscation

and reverse-engineering skills for the attacker to understand this function and to find a valid key.

It is essential not to allow the deobfuscation function to appear in the same procedure or even in the same program as the corresponding obfuscation function. Otherwise, the attacker can use them to compromise the license checking function by using it to deobfuscate its input constants and its intermediate results. This analysis would quickly reveal the identity of the obfuscated operators, putting this attacker in essentially the same position as the attacker who is given debugger access to the unobfuscated function of Fig 6.1.

Since there is no known obfuscation function for division by a variable that has been

obfuscated in residue arithmetic, it is not possible to use this method to obfuscate this arithmetic operation. For example, if we obfuscate the program in Fig. 6.3 and obtain the obfuscated program in Fig. 6.4, we have not significantly increased the difficulty of obtaining its license keys, "x" and "y". It is easy for an adversary to attack the obfuscated program. If the attacker sets a break point at "int t = deobf(te)", he will find a way to call the deobfuscation function on an arbitrary argument. Deobfuscation in the statement "int y = deobf(ye);" is necessary, since we have no function available for division by a variable. By contrast, the function "obfMod99" is available for the obfuscation (in Fig. 6.2) of the function of Fig. 6.1, since we have a function for division by a constant.

6.2 One-dimensional Homomorphic Obfuscations

There are an infinity of potential coding methods for encoding and decoding, but, in practice, we should choose one that is easy to implement. In this thesis, we consider encoding only integer constants and variables. For integers, there are four common operations: addition, subtraction-, multiplication, and division. In order to encode variables in residue numbers, we propose the following definitions.

6.2.1 Definition of Homomorphic Obfuscations

Definition 26 *(Homomorphic and isomorphic obfuscations) If a function $f : Z/nZ \rightarrow Z/mZ$ satisfies the following condition that*

$$\textit{for any two } [x], [y] \in Z/nZ, \textit{ we have } f([x] + [y]) = f([x]) + f([y]).$$

We call $f : Z/nZ \rightarrow Z/mZ$ a homomorphic obfuscation from Z/nZ to Z/mZ.

If a homomorphic obfuscation from Z/nZ to Z/mZ is also a bijection, we call it an isomorphic obfuscation from Z/nZ to Z/mZ.

From $f([x]+[y]) = f([x]) + f([y]))$, it is easy to prove the following basic results:

1. $f([0]_n) = [0]_m$
2. $f([x]-[y]) = f([x]) - f([y])$
3. $f([x][y]) = f([x])f([y])$, so $f([x]_n) = [x]_m f([1]_n)$

6.2.2 Examples of Homomorphic Obfuscations

1. For any Z/nZ and Z/mZ, there is always a trivial homomorphic obfuscation $f : Z/nZ \to Z/mZ$:

$$f([x]_n) = [0]_m \in Z/mZ, \text{ for any } [x]_n \in Z/nZ$$

2. For any Z/nZ, there is an identity homomorphic obfuscation $f : Z/nZ \to Z/nZ$ as follows:

$$f([x]_n) = [x]_n, \text{ for any } [x]_n \in Z/nZ$$

   In fact, it is an identity isomorphic obfuscation from Z/nZ to itself.

3. For any Z/nZ and $Z/2nZ$, there is a homomorphic obfuscation $f : Z/nZ \to Z/2nZ$ as follows:

$$f([x]_n) = [2x]_{2n} \in Z/2nZ, \text{ for any } [x]_n \in Z/nZ$$

4. For $Z/10Z$ and $Z/5Z$, there is a homomorphic obfuscation $f : Z/10Z \to Z/5Z$ as follows:

$$f([x]_{10}) = [2x]_5 \in Z/5Z, \text{ for any } [x]_{10} \in Z/10Z$$

   that is

$$f([0]_{10}) = [0]_5,\ f([1]_{10}) = [2]_5,\ f([2]_{10}) = [4]_5,\ f([3]_{10}) = [1]_5,\ f([4]_{10}) = [3]_5$$
$$f([5]_{10}) = [0]_5,\ f([6]_{10}) = [2]_5,\ f([7]_{10}) = [4]_5,\ f([9]_{10}) = [1]_5,\ f([9]) = [3]_5.$$

5. For any Z/pZ and Z/qZ in which p, q are two distinct prime numbers, there is only the trivial homomorphic obfuscation from Z/pZ to Z/qZ.

6.2.3 Representation of Homomorphic Obfuscations

Theorem 9 *(First representation theorem for homomorphic obfuscations) For any Z/nZ and Z/mZ, if l is an integer such that $m|nl$, then the function*

$$f([x]_n) = [lx]_m \in Z/mZ, \text{ for any } [x]_n \in Z/nZ$$

is a homomorphic obfuscation from Z/nZ to Z/mZ.

On the other hand, if f is a homomorphic obfuscation from Z/nZ to Z/mZ, then

$$f([x]_n) = [lx]_m \in Z/mZ, \text{ for any } [x]_n \in Z/nZ$$

and

$$m|nl$$

where $[l]_m = f([1])$. Furthermore, there exists at least one l satisfying the above conditions in the range $0 \leq l < m$.

Proof.

Let l be an integer such that $m|nl$; we first prove that the

$$f([x]_n) = [lx]_m \in Z/mZ, \text{ for any } [x]_n \in Z/nZ$$

is really a function. That means, if $[x]_n, [y]_n \in Z/nZ$ and $[x]_n = [y]_n$, we should have $f([x]) = f([y])$, e.g. $[lx]_m = [ly]_m$ in Z/mZ. Since $[x]_n = [y]_n$ we have $n|(x-y)$, so there is an integer d such that $x - y = nd$. As a result, $lx - ly = l(x-y) = lnd = nl \cdot d$. By the assumption $m|nl$, we have $m|nl \cdot d$, so $m|(lx - ly)$; that means $[lx]_m = [ly]_m$

in Z/mZ. It is easy to prove that this function f satisfies $f([x]_n + [y]_n) = f([x]_n) + f([y]_n)$.

On the other hand, if f is a homomorphic obfuscation from Z/nZ to Z/mZ and $[l]_m = f([1])$, from $[n]_n = [0]_n \in Z/nZ$, $[nl]_m = [n]_m[l]_m = [n]_m f([1]_n) = f([n]_n) = f([0]_n) = [0]_m \in Z/mZ$; that is, $m|nl$. By $[(l \pm m)x]_m = [lx]_m$, it is easy to see we can choose an l such that $0 \leq l < m$.

Corollary 1 *For any Z/nZ and Z/mZ, there is only the trivial homomorphic obfuscation from Z/nZ to Z/mZ if and only if n and m are two relatively prime numbers.*

Proof. It is easy to see that n and m are two relatively prime numbers if and only if $m|nl$ means $m|l$. While $m|l$ if and only if $[l]_m = [0]_m$ in Z/mZ, we can conclude our result from the above corollary.

Corollary 2 *For any Z/nZ and an integer l, the following function f is a homomorphic obfuscation from Z/nZ to Z/nZ:*

$$f([x]) = [xl] \text{ for all } [x] \in Z/nZ$$

On the other hand, if a function f from Z/nZ to Z/nZ is a homomorphic obfuscation, then there exists an integer l such that

$$f([x]) = [xl] \text{ for all } [x] \in Z/nZ$$

Corollary 3 *(First representation theorem for isomorphic obfuscations) For any Z/nZ and an integer l such that l and n are relatively prime, then the following homomorphic obfuscation f from Z/nZ to Z/nZ is an isomorphic obfuscation:*

$$f([x]) = [xl] \text{ for all } [x] \in Z/nZ.$$

On the other hand, if a homomorphic obfuscation f from Z/nZ to Z/nZ is an isomorphic obfuscation, then there exists an integer l such that l and n are relatively prime and

$$f([x]) = [xl] \text{ for all } [x] \in Z/nZ$$

Corollary 4 *For any Z/nZ, there are total of $\phi(n)$ isomorphic obfuscations from Z/nZ to Z/nZ, where $\phi(n)$ is the Euler phi function.*

6.3 K-dimensional Homomorphic Obfuscations

In the previous section, we discussed homomorphic obfuscation in a one-dimensional setting. In this section, we extend this concept to multiple-dimensional cases through definitions and representations of homomorphic obfuscations.

6.3.1 Basic Definitions

Definition 27 *(Homomorphic and isomorphic obfuscations) If a function $f : Z/nZ \to Z/m_1Z \times Z/m_2Z \times \ldots \times Z/m_kZ$ satisfies the condition that*

$$\text{for any two } [x], [y] \in Z/nZ, \text{ we have } f([x] + [y]) = f([x]) + f([y]),$$

then we call it a homomorphic obfuscation from Z/nZ to $Z/m_1Z \times Z/m_2Z \times \ldots \times Z/m_kZ$.
If a homomorphic obfuscation from Z/nZ to $Z/m_1Z \times Z/m_2Z \times \ldots \times Z/m_kZ$ is also a bijection, we call it an isomorphic obfuscation from Z/nZ to $Z/m_1Z \times Z/m_2Z \times \ldots \times Z/m_kZ$.

For any Z/nZ and $Z/m_1Z \times Z/m_2Z \times \ldots \times Z/m_kZ$, there is always a trivial homomorphic obfuscation as follows:

$$f([x]_n) = ([0]_{m_1}, [0]_{m_2}, \ldots, [0]_{m_k}) \text{ for any } [x] \in Z/nZ.$$

For a homomorphic obfuscation $f : Z/nZ \to Z/m_1Z \times Z/m_2Z \times \ldots \times Z/m_kZ$ and $[x], [y] \in Z/nZ$, it is easy to prove the following properties:

1. $f([0]_n) = ([0]_{m_1}, [0]_{m_2}, \ldots, [0]_{m_k})$.

2. $f([x] - [y]) = f([x]) - f([y])$.

3. $f([x][y]) = f([x])f([y])$, so $f([x]_n) = ([x]_{m_1}, [x]_{m_2}, \ldots, [x]_{m_k})f([1]_n)$.

6.3.2 Representation of Homomorphic Obfuscations

Theorem 10 *(The first representation theorem for homomorphic obfuscations) For any Z/nZ and $Z/m_1Z \times Z/m_2Z \times \ldots \times Z/m_kZ$, if $l_1, l_2, \ldots, l_k$ are integers such that $m_i | nl_i$, for $i = 1, 2, \ldots, k$, then the function*

$$f([x]_n) = ([l_1x]_{m_1}, [l_2x]_{m_2}, \ldots, [l_kx]_{m_k}), \text{ for any } [x]_n \in Z/nZ$$

is a homomorphic obfuscation from Z/nZ to $Z/m_1Z \times Z/m_2Z \times \ldots \times Z/m_kZ$.
On the other hand, if f is a homomorphic obfuscation from Z/nZ to $Z/m_1Z \times Z/m_2Z \times \ldots \times Z/m_kZ$, then

$f([x]_n) = ([l_1x]_{m_1}, [l_2x]_{m_2}, \ldots, [l_kx]_{m_k})$, for any $[x]_n \in Z/nZ$ and

$m_i | nl_i$, for $i = 1, 2, \ldots, k$ where $([l_1]_{m_1}, [l_2]_{m_2}, \ldots, [l_k]_{m_k}) = f([1])$.

Furthermore, we can choose these integers such that $0 \leq l_i < m_i$ for $i = 1, 2, \ldots, k$, and we say the homomorphic obfuscation f has the representation $(l_1, l_2, \ldots, l_k)$.

Corollary 5 *For any Z/nZ and $Z/m_1Z \times Z/m_2Z \times \ldots \times Z/m_kZ$, there is only the trivial homomorphic obfuscation from Z/nZ to $Z/m_1Z \times Z/m_2Z \times \ldots \times Z/m_kZ$ if and only if, for $i = 1, 2, \ldots, k$, n and m_i are two relatively prime numbers.*

Corollary 6 *Assuming that Z/nZ and $Z/m_1Z \times Z/m_2Z \times \ldots \times Z/m_kZ$ satisfy $n = m_1 \times m_2 \times \ldots \times m_k$, we have the following results:*

For any integers $l_1, l_2, \ldots, l_k$, the function

$$f([x]_n) = ([l_1x]_{m_1}, [l_2x]_{m_2}, \ldots, [l_kx]_{m_k}), \text{ for any } [x]_n \in Z/nZ$$

is a homomorphic obfuscation from Z/nZ *to* $Z/m_1Z \times Z/m_2Z \times \ldots \times Z/m_kZ$.

On the other hand, if f *is a homomorphic obfuscation from* Z/nZ *to* $Z/m_1Z \times Z/m_2Z \times \ldots \times Z/m_kZ$, *then*

$$f([x]_n) = ([l_1x]_{m_1}, [l_2x]_{m_2}, \ldots, [l_kx]_{m_k}), \text{ for any } [x]_n \in Z/nZ$$

where $([l_1]_{m_1}, [l_2]_{m_2}, \ldots, [l_k]_{m_k}) = f([1])$.

Theorem 11 *(The second representation theorem for homomorphic obfuscations) Assume that* Z/nZ *and that* $Z/m_1Z \times Z/m_2Z \times \ldots \times Z/m_kZ$ *satisfy that* $m_1, m_2, \ldots, m_k \in Z$ *are pairwise relatively prime and* $n = m_1 \times m_2 \times \ldots \times m_k$, *we have the following results:*

For any integer l, *then the function*

$$f([x]_n) = ([lx]_{m_1}, [lx]_{m_2}, \ldots, [lx]_{m_k}), \text{ for any } [x]_n \in Z/nZ$$

is a homomorphic obfuscation from Z/nZ *to* $Z/m_1Z \times Z/m_2Z \times \ldots \times Z/m_kZ$.

On the other hand, if f *is a homomorphic obfuscation from* Z/nZ *to* $Z/m_1Z \times Z/m_2Z \times \ldots \times Z/m_kZ$, *then there exists an integer* l *such that*

$$f([x]_n) = ([lx]_{m_1}, [lx]_{m_2}, \ldots, [lx]_{m_k}), \text{ for any } [x]_n \in Z/nZ.$$

Proof. We need to prove only the second part of our result. By Corollary 6, for a homomorphic obfuscation f from Z/nZ to $Z/m_1Z \times Z/m_2Z \times \ldots \times Z/m_kZ$, there are integers $l_1, l_2, \ldots, l_k$ such that

$$f([x]_n) = ([l_1x]_{m_1}, [l_2x]_{m_2}, \ldots, [l_kx]_{m_k}), \text{ for any } [x]_n \in Z/nZ.$$

By Theorem A.28 in [41, page 255], there is an integer l such that $[l]_{m_i} = [l_i]_{m_i}$, for $i = 1, 2, \ldots, k$. Therefore, for any x, $[lx]_{m_i} = [l_ix]_{m_i}$, for $i = 1, 2, \ldots, k$. We get $f([x]_n) = ([lx]_{m_1}, [lx]_{m_2}, \ldots, [lx]_{m_k})$, for any $[x]_n \in Z/nZ$.

Theorem 12 *(The representation theorem for isomorphic obfuscations) Assuming that* Z/nZ *and that*$Z/m_1Z \times Z/m_2Z \times \ldots \times Z/m_kZ$ *satisfy* $n = m_1 \times m_2 \times \ldots \times m_k$ *and* $m_1, m_2, \ldots, m_k \in Z$ *are pairwise relatively prime integers, we have the following results:*

For any integer l *such that* l *and* n *are relatively prime, the function*

$$f([x]_n) = ([lx]_{m_1}, [lx]_{m_2}, \ldots, [lx]_{m_k}), \text{ for any } [x]_n \in Z/nZ$$

is an isomorphic obfuscation from Z/nZ *to* $Z/m_1Z \times Z/m_2Z \times \ldots \times Z/m_kZ$.

On the other hand, if f *is an isomorphic obfuscation from* Z/nZ *to* $Z/m_1Z \times Z/m_2Z \times \ldots \times Z/m_kZ$, *then there exists an integer* l *such that*

$$f([x]_n) = ([lx]_{m_1}, [lx]_{m_2}, \ldots, [lx]_{m_k}), \text{ for any } [x]_n \in Z/nZ.$$

Furthermore, $([l]_{m_1}, [l]_{m_2}, \ldots, [l]_{m_k}) = f([1])$ *and* l *and* n *are relatively prime.*

Proof. By Theorem 10, for any integer l satisfying that l and n are relatively prime, the function $f([x]_n) = ([lx]_{m_1}, [lx]_{m_2}, \ldots, [lx]_{m_k})$, for any $[x]_n \in Z/nZ$ is a homomorphic obfuscation from Z/nZ to $Z/m_1Z \times Z/m_2Z \times \ldots \times Z/m_kZ$. We prove this function is also a bijection. Firstly, we prove that

$$\text{if } f([x]_n)=([0]_{m_1}, [0]_{m_2}, \ldots, [0]_{m_k}), \text{ then } [x]_n = [0]_n.$$

In fact, if $f([x]_n) = ([0]_{m_1}, [0]_{m_2}, \ldots, [0]_{m_k})$, then $([lx]_{m_1}, [lx]_{m_2}, \ldots, [lx]_{m_k}) = ([0]_{m_1}, [0]_{m_2}, \ldots, [0]_{m_k})$, that means $m_i|lx$ for $i = 1, 2, \ldots, k$. Because m_1, m_2, ..., $m_k \in Z$ are pairwise relatively prime integers, we have $m_1m_2 \ldots m_k|lx$. By $n = m_1 \times m_2 \times \ldots \times m_k$ and l and n are relatively prime, we have $n|x$; that means $[x]_n = [0]_n$. Secondly, by the Property 2 of homomorphic obfuscations, $f([x] - [y]) = f([x]) - f([y]))$, we have: if $[x]_n = [y]_n$, then $f([x]_n) = f([y]_n)$. Because Z/nZ and $Z/m_1Z \times Z/m_2Z \times \ldots \times Z/m_kZ$ are finite sets with the same cardinality, the above function f must be a bijection, so it is an isomorphic obfuscation.

On the other hand, if a function f from Z/nZ to $Z/m_1Z \times Z/m_2Z \times \ldots \times Z/m_kZ$ is an isomorphic obfuscation, by Theorem 11, we have an integer l such that $f([x]_n) = ([lx]_{m_1}, [lx]_{m_2}, \ldots, [lx]_{m_k})$, for any $[x]_n \in Z/nZ$.

We prove l and n are relatively prime. Otherwise, l and n have a common divisor $d > 1$. Let $n = n'd$ and $l = l'd$, then $[n']_n \neq [0]_n$, but $[ln']_{m_i} = [l'd \times n']_{m_i} = [l' \times n'd]_{m_i} = [l' \times n]_{m_i} = [0]_{m_i}$, for $i = 1, 2, \ldots, k$; that is $f([n']_n) = ([0]_{m_1}, [0]_{m_2}, \ldots, [0]_{m_k})$. This concludes our result.

6.4 Homomorphic Functions and Orders

Definition 28 *For a homomorphic obfuscation $f : Z/nZ \rightarrow Z/mZ$ and the natural orders of Z/nZ and Z/mZ, if for any $[x], [y] \in Z/nZ$, $[x] \leq [y]$ implies $f([x]) \leq f([y])$, we call it an order-keeping homomorphic obfuscation.*

Clearly, the trivial homomorphic obfuscation from Z/nZ to Z/mZ is an order-keeping homomorphic obfuscation, but not all homomorphic obfuscations are order-keeping homomorphic obfuscations. For example, $f : Z/5Z \rightarrow Z/10Z$,

$$f([x]_5) = [4x]_{10} \in Z/10Z, \text{ for any } [x]_5 \in Z/5Z.$$

that is

$$f([0]_5) = [0]_{10}, f([1]_5) = [4]_{10}, f([2]_5) = [8]_{10}, f([3]_5) = [2]_{10}, f([4]_5) = [6]_{10}.$$

The above function is a homomorphic obfuscation from $Z/5Z$ to $Z/10Z$, but it does not keep the natural orders of $Z/5Z$ and $Z/10Z$.

Theorem 13 *(First characteristic theorem for order-keeping homomorphic obfuscations) For a homomorphic obfuscation f from Z/nZ to Z/mZ, represented as*

$$f([x]_n) = [lx]_m \in Z/mZ, \text{ for any } [x]_n \in Z/nZ$$

where $[l]_m = f([1])$ *and* $0 \leq l < m$ *according to Theorem 9, then* f *is an order-keeping homomorphic obfuscation if and only if* $0 \leq (n-1)l < m$.

Corollary 7 *If* $n > m$, *there is no order-keeping homomorphic obfuscation from* Z/nZ *to* Z/mZ *other than the trivial homomorphic obfuscation.*

Corollary 8 *For any* Z/nZ, *there are only two order-keeping homomorphic obfuscations from* Z/nZ *to* Z/nZ, *the trivial one and the identity homomorphic obfuscations.*

6.5 Division of Integers by a Constant

The division of residue numbers is a complicated problem. In line 38 and 39 of a patent [21, page 17], Chow et al. wrote in [21] "Most texts like [62] also indicate that division is impossible." In reality, we can not say division of residue numbers is impossible. Knuth commented on the division of residue numbers, writing "It is even more difficult to perform division" in line 21 of pp. 285; see also Exercise 4.3.2-11 at pp. 293 [62]. Chow et al.'s patent [21, page 17] describes a method for division of residue numbers, at line 39 and 40, "However, the invention provides a manner of division by a constant." We assert and will prove that this method is invalid. The main problem is that the equations (19) and (20) in the patent [21, page 17] are incorrect. Because these two equations are the basis for the techniques of the division of residue numbers, the invention as described in [21, page 17] is completely invalid. The solution to division by a constant d in the patent [21] has another problem – an overly restrictive condition that such d can be only one of its bases.

In Sections 6.2 and 6.3 of this chapter, we have laid a sound grounding for constructing a mechanism for division by constants. Firstly, we give a solution to division by a constant d. Assume that Z/nZ and $Z/m_1Z \times Z/m_2Z \times \ldots \times Z/m_kZ$ satisfy $n = m_1 \times m_2 \times \ldots \times m_k$ and $m_1, m_2, \ldots, m_k$ are pairwise relatively prime integers

and l and n are relatively prime. We define an isomorphic obfuscation f from Z/nZ to $Z/m_1Z \times Z/m_2Z \times \ldots \times Z/m_kZ$ as follows:

$$f([y]_n) = ([dy]_{m_1}, [dy]_{m_2}, \ldots, [dy]_{m_k}), \text{ for any } [y]_n \in Z/nZ.$$

Then, for any $0 \leq x < n$ such that $d|x$, we have

$$f([\frac{x}{d}]) = ([x]_{m_1}, [x]_{m_2}, \ldots, [x]_{m_k}).$$

This solution is simple and easy to implement, and it is better than that in Patent [21]. As for the following restrictions to our solution, we make the following comments.

1. Our first constraint is that $d|x$. This is unsurprising, for if this constraint is not met, then
$$\frac{x}{d}$$
is not a linear function of x.

2. Our second constraint is that d and n are relatively prime. This is not problematic in some applications, for when obfuscating a program containing a single constant d, we have the freedom to choose n so that this condition is satisfied.

In the case that d is one of $m_1, m_2, \ldots, m_k$, as in Chow et al.'s patent [21], we see some fundamental difficulties, for we can prove the following result.

Theorem 14

$[\frac{y}{m_i}]_{m_i}$ is not a linear function of $[y]_{m_1}, [y]_{m_2}, \ldots, [y]_{m_k}$; that is, there are no constants with respect to y, $c_1, c_2, \ldots, c_k$, such that

$$[\frac{y}{m_i}]_{m_i} = c_1[y]_{m_1} + c_2[y]_{m_2} + \ldots + c_k[y]_{m_k} \tag{6.1}$$

where $m_i|y$.

Let us first consider the case $k = 2$ and $i = 1$ and $m_1 < m_2$. If the above linear function exists, then

$$[\frac{y}{m_1}]_{m_1} = c_1[y]_{m_1} + c_2[y]_{m_2} \tag{6.2}$$

holds for all $m_1|y$. Let $y = m_1^2$; we have

$$[\frac{m_1^2}{m_1}]_{m_1} = c_1[m_1^2]_{m_1} + c_2[m_1^2]_{m_2}. \tag{6.3}$$

So, $c_2[m_1^2]_{m_2} = 0$. For m_1 and m_2 are relatively prime, $c_2 = 0$.

Now Equation (6.2) is reduced to

$$[\frac{y}{m_1}]_{m_1} = c_1[y]_{m_1}. \tag{6.4}$$

Let $y = m_1 m_2$, we have $[m_2]_{m_1} = 0$. This is a contradiction. For other cases, the proof is similar.

Compared with the method in Chow et al.'s patent [21], where d can be only one of $m_1, m_2, \ldots, m_k$, our solution has more freedom for choosing d. When d is one of $m_1, m_2, \ldots, m_k$, our solution does not work, but we can choose another set of parameters $m_1, m_2, \ldots, m_k$ so that our solution works. In fact, the solution in Chow et al.'s patent [21] itself does not really work for the case that d is one of $m_1, m_2, \ldots, m_k$.

6.6 Division of Integers by Several Constants

Assume that Z/nZ and $Z/m_1Z \times Z/m_2Z \times \ldots \times Z/m_kZ$ satisfy $n = m_1 \times m_2 \times \ldots \times m_k$ and $m_1, m_2, \ldots, m_k$ are pairwise relatively prime integers, $d_1, d_2, \ldots, d_p$ are p integers, and d_i and n are relatively prime for any $i = 1, 2, \ldots, p$. Let $d = d_1 d_2 \ldots d_p$; we define an isomorphic obfuscation f from Z/nZ to $Z/m_1Z \times Z/m_2Z \times \ldots \times Z/m_kZ$ as follows:

$$f([y]_n) = ([dy]_{m_1}, [dy]_{m_2}, \ldots, [dy]_{m_k}), \text{ for any } [y]_n \in Z/nZ$$

Then, for any $1 \leq i \leq p$ and $0 \leq x < n$ such that $d_i|x$, denote $d/d_i = d_1 \cdots d_{i-1}d_{i+1} \cdots d_k$ as d'_i, we have

$$f([\frac{x}{d_i}]) = ([d'_i x]_{m_1}, [d'_i x]_{m_2}, \ldots, [d'_i x]_{m_k}).$$

6.7 Conclusion

While residue number coding can be used in RSA cryptography, its applications to software obfuscation to encode variables to hide the real meaning of these variables as said in patent [21] has a big flaw. The solution to division of residue numbers proposed in this thesis is based on a sound grounding in number theory and can be used in software obfuscation to hide integers. This result is new in the software obfuscation literature.

The method proposed in this chapter, an improvement to Chow's approach, is used to hide important constants and operations in programs. The impossibility results [10] by Barak et al. reveal that programs cannot be securely concealed would greatly increase the difficulties faced by a reverse engineer. the method developed in this chapter or any other obfuscation techniques. But, especially if combined with other techniques, our method might help protect program from reverse engineering.

Some points in our methods require further exploration, such as the security of our methods and the combination of our methods with other software obfuscation techniques. These are topics for future research. In next chapter, we will further apply homomorphic function to obfuscate data structures.

Chapter 7

Application of Homomorphic Function

In the previous chapter, we established a sound ground for residue number coding for software obfuscation. Especially, we used this to develop an algorithm for division by several constants, correcting an error in an earlier publication [21]. In this chapter, we describe further applications of homomorphic functions to software obfuscation [173, 174].

The remainder of this chapter is structured as follows. In Section 7.1 we describe array transformation, an important technique in software obfuscation. In Section 7.2, the focus of this chapter, we apply homomorphic functions to array transformations. This chapter concludes in section 7.3.

7.1 Array Transformations

Data transformation is one of several algorithms used in software obfuscation [33]. Array index change, array folding, and array flattening are some data transformations. As said in [33], by adding data complexity into the program, array index change, array

folding and flattening make a program more difficult to understand and to reverse engineer.

7.1.1 Array Index Change

The following Fig. 7.1 is a simple example of the array index change method [33]:

```
int A[9];            int A1[5], A2[4];
A[i] = ...           if((i%2 == 0))
                ⇒          A1[i/2] =...
                    else
                          A2[i/2] = ...
```

Figure 7.1: An example of array index change

7.1.2 Array Folding and Flattening

Array folding increases the dimension of an array in the code, such as transforming a two-dimensional array to a one-dimensional array. In contrast, array flattening decreases the dimension of an array in the code, such as transforming a two-dimensional array to a four-dimensional array.

7.2 Application of Homomorphic Function to Arrays

Homomorphic functions can be used to obfuscate programs through changing the index or the dimension of an array in the programs to obfuscate them. In the following, we describe in detail four methods, which we reported in [173, 174], to apply homomorphic functions to software obfuscation.

7.2.1 Index Change

Homomorphic functions can change the index of an array in software to obfuscate it. For an array A[n], the procedure has two steps as follows:

1. Find an m such that $m > n$, and n and m are relatively prime.
2. Change the array into another array B[n], and the element A[i] is turned into b[i*m mod n].

The homomorphic function f: $Z/nZ \rightarrow Z/nZ$ defined by $f([i]_n) = [i * m]_n$ is an isomorphism, thus the above replacement guarantees the semantics of the original program.

Example 28 (Index change:) *For the program in Fig. 7.2, we choose* $m = 3$. *The obfuscated program is in Fig. 7.3.*

We cannot use this index change method to arrays of real numbers, since round-off errors may occur in some operations of floating numbers. A round-off error is the difference between the calculated approximation of a number and its exact value in mathematical meaning. It occurs in a computation by rounding results at one or more intermediate steps. The other common cases of round-off errors happen when two quantities very close to each other are subtracted, a number is divided by a number which is close to zero, and a big number and a small one are added. This index change method can be applied safely only to sequences of array operations which are fully commutative.

7.2.2 Index and Dimension Change

Homomorphic functions can be used to change the index and the length of dimension of an array in programs to obfuscate them. To achieve this, for an array A[n], we has the following procedure:

```
...
int A[100];
...;
S = 0;
for(i = 0; i < 100; i++)
  S = S + A[i];
...
```

Figure 7.2: An unobfuscated program

```
...
int B[100];
...
S = 0;
for(i = 0; i < 100; i++)
    S = S + B[i*3 mod 100];
...
```

Figure 7.3: An obfuscated version of the program in Fig. 7.2

1. Find an m such that $m > n$, and n and m are relatively prime.

2. Change the array into another array B[m] and the element A[i] is turned into b[i*n mod m].

The above method can be regarded as two steps. Firstly, extend array A[n] to C[m] with C[i] = A[i] for $0 \leq i < n$ and C[i] undefined for $n \leq i < m$]. Then, change the array C[m] into b[m] by replacing C[i] with B[i*n mod m] as in the index change method.

Example 29 (Index and dimension change:) *For the program in Fig. 7.4, we choose* $m = 101$. *The obfuscated program is in Fig. 7.5.*

```
...
int A[100];
...;
S = 0;
for(i = 0; i < 100; i++)
  S = S + A[i];
...
```

Figure 7.4: An unobfuscated program

```
...
int B[101];
...
S = 0;
for(i = 0; i < 100; i++)
    S = S + B[i*100 mod 101];
...
```

Figure 7.5: An obfuscated version of the program in Fig. 7.4

7.2.3 Array Folding

Homomorphic functions can increase the number of dimensions of an array in software to obfuscate it. The technique is called array folding. For an array A[n], we assume $n > 2$. The array folding procedure is as follows:

1. If n is a prime, let $m = n + 1$; otherwise $m = n$.
2. Extend A[n] into C[m] by C[i] = A[i] for $0 \leq i < n$ and C[i] undefined for $n \leq i < m$.
3. Factor m into m_1 and m_2. Replace C[m] with B[m_1, m_2] through B[i mod m_1, i mod m_2] = C[i] for $0 \leq i < m$.
4. Replace any A[i] with B[i mod m_1, i mod m_2] in the unobfuscated program.

Example 30 (Array folding:) *For the program in Fig. 7.6, we choose* $m = 3$.

1. *Factor 100 into 4 × 25, two relatively prime integers.*
2. *Turn the one-dimensional array A[100] into a two-dimensional array B[4, 25] and let B[i mod 4, i mod 25] = A[i] for* $0 \leq i < 100$.
3. *Replace any A[i] with B[i mod 4, i mod 25] in the unobfuscated program.*

The obfuscated program is in Fig. 7.7.

```
...
S = 0;
for(i = 0; i < 100; i++)
  S = S + A[i];
...
```

Figure 7.6: An unobfuscated program

```
...
S = 0;
for(i = 0; i < 4; i++)
  for(j = 0; i < 25; j++)
    S = S + B[i, j];
...
```

Figure 7.7: An obfuscated version of the program in Fig. 7.6

7.2.4 Array Flattening

Homomorphic functions can be used to decrease the number of dimension of an array in software to obfuscate it and is called array flattening. For a two-dimensional array $A[n_1, n_2]$, the array flattening procedure is as follows:

1. Find two relatively prime integers m_1 and m_2 such that $n_1 \leq m_1$ and $n_2 \leq m_2$. Let $m = m_1 * m_2$.

2. Turn the two-dimensional array $A[n_1, n_2]$ into another two-dimensional array $C[m_1, m_2]$ by $C[i, j] = A[i, j]$ for $0 \leq i < m_1$ and $0 \leq j < m_2$, and $C[i, j]$ undefined otherwise. Replace all $A[i, j]$ with $C[i, j]$.

3. Find two relatively prime integers k_1 and k_2 such that $k_1 * m_1 + k_2 * m_2 = 1$.

4. Turn the two-dimensional array $C[n_1, n_2]$ into a one-dimensional array $B[m]$ and let $B[i] = C[i \ mod \ m_1, i \ mod \ m_2]$ for $0 \leq i < m$.

5. Replace any $A[i, j]$ with $B[(i * k_1 + j * k_2) \ mod \ m]$ for $0 \leq i < n_1$ and $0 \leq j < n_2$ in the unobfuscated program.

Example 31 (Array flattening) *For the program in Fig. 7.8,*

1. *We choose 4 and 27.*

2. *Time 4 and 27 into 108*

3. *Turn the two-dimensional array B[4, 26] into a one-dimensional array A[108] and let*

   - *A[i] = B[i mod 4, i mod 27] for* $0 \leq i < 104$.
   - *A[i] = any value for* $104 \leq i < 108$.

4. *Replace any B[i mod 4, i mod 27] with A[i] for* $0 \leq i < 104$ *in the unobfuscated program.*

The new obfuscated program is as in Fig. 7.9.

```
...
int B[4, 26];
...
S = 0;
for(i = 0; i < 4; i++)
  for(j = 0; i < 26; j++)
    S = S + B[i, j];
...
```

Figure 7.8: An unobfuscated program

```
...
int A[108];
...
S = 0;
for(i = 0; i < 104; i++)
  S = S + A[i];
...
```

Figure 7.9: An obfuscated version of the program in Fig. 7.8

7.3 Conclusions

Array transformation is an important method in software obfuscation. This chapter concerns applications of the residue number coding to obfuscate data structures in software. Because of round-off errors, array transformation methods developed in this chapter can be applied to only operations which are fully commutative. We will investigate more applications of homomorphic functions to software obfuscation in our future research.

This chapter concludes software obfuscation. Part III contains our conclusions and area for future research.

Part III

Conclusions and Future Work

Chapter 8

Conclusions and Future Work

8.1 Conclusions

Computer and communication industries develop so rapidly that the demand for software becomes larger and larger and the demand for software protections, such as copyright and anti-tampering defense, are becoming more important to software users and developers. Software watermarking and obfuscation are two software-based techniques to protect software from piracy and tampering. In this thesis, we study software watermarking and obfuscation. Firstly, we review the current security issues in the software community. Then a detailed survey of software watermarking to describe aims, problems, and techniques in this field is given. The main contributions of this thesis is to formalize two important concepts in software watermarking (extraction and recognition), to propose an algorithm for software watermarking, and to establish a technique for software obfuscation.

8.1.1 Software Watermarking

In order to trace actions of unauthorized use, modification, and tampering, software watermarking inserts a piece of secret information into the protected software. The developer can use this secret message to prove a claim to the ownership of this software if piracy occurs. Since this research subject is a relatively new one, many core concepts in it are informal. Some terms have different meanings in different literatures. As a result, there is a need to formalize several key concepts which are essential for progress in this area.

In this thesis, we formalized two core concepts in software watermarking: extraction and recognition. For extraction, we defined important concepts such as watermark, embedding, candidate watermark, extractable embedding, and representatively extractable embedding. A software watermark is the bits of information we try to insert into a program. An embedding algorithm is a process that inserts a watermark into a program and makes the new program still executable. A candidate watermark is the bits of information we can really insert into a program through an embedding algorithm. An extractable embedding is such an algorithm for software watermarking that the watermark inserted by this algorithm can extracted by some algorithm. For a representatively extractable embedding, not all watermarks inserted by such an algorithm can be extracted. We proved that not all embeddings are an extractable embedding, but all are representatively extractable.

For recognition, in addition to the concepts of watermark, embedding, candidate watermark, we defined important concepts such as positive-partial recognition, negative-partial recognition, recognition, and the strength of recognition. A positive-partial recognition will report the existence of a watermark in a program if this watermark is really in this program, but a positive-partial recognition might say a watermark exists in a program even though this watermark is actually not in this

program. On the contrary, a negative-partial recognition will report the nonexistence of a watermark in a program if this watermark is not in this program. A positive-partial recognition might report the nonexistence of a watermark in a program even though this watermark is actually in this program. Given an embedding algorithm, we prove the existence and uniqueness of its corresponding recognition algorithm. We also proved the existence of the weakest positive-partial recognition and the strongest positive-partial recognition. This result is also valid for negative-partial recognitions.

We present an algorithm for software watermarking. It embeds a watermark into a program through the register allocation graph of this program. When developing this algorithm, we pointed out one flaw in previous work in this field. It is in this algorithm that we found the importance of the concept such as extractable embedding and representatively extractable embedding.

Through the work mentioned above, we have established a sound mathematical foundation for extraction and recognition, two important concepts in the software watermarking process.

8.1.2 Software Obfuscation

Software obfuscation transforms a program into another one which is hard for people and software analysis tools to understand. In this thesis, we also studied software obfuscation since it is closely related with software watermarking. Additionally, software obfuscation is a technique to protect software watermark from detection and attack.

After a brief survey of software obfuscation, we propose a technique to obfuscate variables and data structures in software. This approach is based on a sound theory called homomorphic function we established in this thesis. This is one of main contributions of this thesis.

Chow et al. used residue number technique to software obfuscation by encoding variables in the original program to hide the true meaning of these variables [21]. But the technique proposed in their work is not correct. In order to establish a sound ground for this type of issue, we used homomorphic functions to hide addition, subtraction, and multiplication in integer operations. With a suitable choice, this method can also apply to division operation. Through examples, we show it is costly to hide comparisons in integer operations.

Data structures are important components of programme and they are key clues for people to understand codes. Obfuscating data structures of programme will make it very hard for an enemy to attack them. The homomorphic functions are also used to obfuscate the data structures of software. We proposed several methods for different cases in data structure obfuscation such as array index change, array index and dimension change, array folding, and array flattening. These methods strengthened existing techniques in data structure obfuscation.

In summary, we corrected the obfuscation method proposed by Chow et al. and applied it to obfuscating integer arrays.

8.1.3 Complexity and Security

In this thesis, all the embedding, extraction, and recognition algorithms connected with the QP algorithm are all of linear time complexity.

As shown by Barak et al., there are no totally secured software watermarking and obfuscation algorithms, so we should combined our methods developed in this thesis with other techniques such as hardware security methods. Though, from Barak et al.'s result, there does not exist any one-way software watermarking function, different situations in real world require different kinds of security. Our thesis aims to construct software watermarking algorithms with linear complexity and obfuscators to hide

important constants, arithmetic operations, and data structures in programs.

8.2 Future Work

Many issues in software watermarking and obfuscation need further research. Firstly, though Barak et al. presented negative results about a universal obfuscater, suitable software obfuscation models for special settings is central for researchers to develop new techniques and evaluate these approaches. Secondly, more work is needed to formalize the core concepts in software watermarking and obfuscation. It will certainly benefit the development of this research area. In the future, we will study the following issues in software watermarking and obfuscation.

8.2.1 Combination of Software Watermarking and Obfuscation

As we know software watermarking and software obfuscation are two closely related branches in software security. Especially, software obfuscation is one of the techniques to protect software watermark. Therefore, how to combine these two methods into a software watermarking system is an important research issue for future work. For example, we will study developing algorithms to insert software watermark into software obfuscation algorithms. This algorithm will be an integrated algorithm for software watermarking and obfuscation.

8.2.2 Applications of Rough Set Theory to Software Security

Rough set theory is a method proposed by Pawlak in 1982 for data mining [104]. The characteristic advantage of rough set method is that it does not need any additional information about data, such as probability in statistics or grade of membership in fuzzy set theory. Many examples of applications of the rough set method to process control, economics, medical diagnosis, biochemistry, environmental science, biology,

chemistry psychology, conflict analysis and other fields are found in [6, 42, 54, 60, 99, 107, 108, 109, 130, 157]. It has also been applied to software security [145, 57] and digital watermarking [56]. As software watermarking is a branch of digital watermarking, we hope rough set theory can be applied to software watermarking, especially to detection and recognition of software watermark.

We have already done some research on rough set theory and attempted to apply it to security problems [158, 159, 160, 161, 162, 163, 164, 167, 168, 169, 170, 175, 176, 177, 178, 182, 183, 184, 185, 186]. We hope we can find more applications of this theory to software watermarking and obfuscation.

Chapter 9

References

Bibliography

[1] B. Adida and D. Wikstrm, "Obfuscated ciphertext mixing," in *Cryptology ePrint Archive, Report 2005/394*, 2005, http://eprint.iacr.org/.

[2] R. Agrawal, P. Haas, and J. Kiernan, "Potpourri: A system for watermarking relational databases," in *the 2003 ACM SIGMOD international conference on Management of data*, June 2003.

[3] R. Agrawal, P. Haas, and J. Kiernan, "Watermarking relational data: framework, algorithms and analysis," *The International Journal on Very Large Data Bases*, vol. 12, no. 2, Aug 2003.

[4] B. Anckaert, B. D. Sutter, D. Chanet, and K. D. Bosschere, "Steganography for executables and code transformation signatures," in *ICISC 2004*, ser. LNCS, vol. 3506, 2005, pp. 425–439.

[5] R. Anderson and A. Petitocolas, "On the limits of steganography," *Public Key Encryption '99, Lecture Notes in Computer Science*, May 1998.

[6] F. Angiulli and C. Pizzuti, "Outlier mining in large high-dimensional data sets," *IEEE Trans. On Knowledge and Data Engineering*, vol. 17, no. 2, pp. 203–215, 2005.

[7] G. Arboit, "A method for watermarking java programs via opaque predicates," in *The Fifth International Conference on Electronic Commerce Research (ICECR-5)*, 2002. [Online]. Available: http://citeseer.nj.nec.com/arboit02method.html

[8] M. Atallah, V. Raskin, C. hempelmann, M. Karahan, R. Sion, K. Triezenberg, and U. Topkara, "Natural language watermarking and tamperproofing," ser. LNCS, vol. 2578, 2003, pp. 196–212.

[9] D. Aucsmith, "Tamper resistant software: an implementation," in *LNCS 1174*, 1996, pp. 317–333.

[10] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang, "On the (im)possibility of obfuscating programs (extended abstract)," in *Advances in Cryptology - CRYPTO 2001*, ser. LNCS, J. Kilian, Ed., vol. 2139, 2001, pp. 1–18.

[11] M. Bobeica, J.-P. Jeral, and CliveGarciaBest, "A study of "root causes of conflict" using latent semantic analysis," in *ISI 2005*, ser. LNCS, vol. 3495, 2005, pp. 595–596.

[12] Z. Bonikowski, E. Bryniarski, and U. Wybraniec-Skardowska, "Extensions and intentions in the rough set theory," *Information Sciences*, vol. 107, pp. 149–167, 1998.

[13] Z. Bonikowski, "Algebraic structures of rough sets," in *Rough Sets, Fuzzy Sets and Knowledge Discovery*, W. Ziarko, Ed. Springer, 1994, pp. 243–247.

[14] D. Brewer and M. Nash, "The chinese wall security policy," in *IEEE Symposium on Security and Privacy*, 1989, pp. 206–214.

[15] E. Bryniaski, "A calculus of rough sets of the first order," *Bull. Pol. Acad. Sci.*, vol. 36, no. 16, pp. 71–77, 1989.

[16] BSA and IDC, "Bsa and idc global software piracy study," in *http://www.bsa.org/usa/research/*, Nov. 22, 2004.

[17] J. Burridge, "Information preserving statistical obfuscation," *STATISTICS AND COMPUTING*, vol. 13, no. 4, pp. 321–327, Oct 2003.

[18] G. Cattaneo, "Abstract approximation spaces for rough theorie," in *Rough Sets in Knowledge Discovery 1: Methodology and Applications*, 1998, pp. 59–98.

[19] J. Chan and W. Yang, "Advanced obfuscation techniques for java bytecode," *JOURNAL OF SYSTEMS AND SOFTWARE*, vol. 71, no. 1, pp. 1–10, Apr. 2004.

[20] H. Chang and M. Atallah, "Protecting software code by guards," in *ACM Workshop Security and Privacy in Digital Rights Management.* ACM Press, 2001, pp. 160–175.

[21] Chow and et al, "Tamper resistant software encoding," *US patent*, vol. 6594761, pp. 1–32, Oct. 2003.

[22] S. Chow, Y. Gu, H. Johnson, and V. A. Zakharov, "An approach to the obfuscation of control-flow of sequential computer programs," in *ISC 2001*, ser. LNCS, vol. 2200, 2001, pp. 144–155.

[23] S. Cimato, A. D. Santis, and U. F. Petrillo, "Overcoming the obfuscation of java programs by identifier renaming," *The Journalof Systems and Software*, 2005.

[24] 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.

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

[26] C. Collberg, G. Myles, and A. Huntwork, "Sandmark–a tool for software protection research," *IEEE Security and Privacy*, vol. 1, no. 4, pp. 40–49, 2003.

[27] 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. [Online]. Available: http://www.cs.arizona.edu/people/kobourov/papers.html

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

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

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

[31] 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.

[32] C. Collberg, C. Thomborson, and D. Low, "Manufacturing cheap, resilient, and stealthy opaque constructs," in *POPL'98*, 1998, pp. 184–196.

[33] C. Collberg, C. Thomborson, and D. Low, "A taxonomy of obfuscating transformations," in *Tech. Report, No.148, Dept. of Computer Sciences, Univ. of Auckland*, 1997.

[34] Stephen A. COOK, "AN OVERVIEW OF COMPUTATIONAL COMPLEXITY," *Communications of the ACM*, Vol. 26, No. 6, 1983, pp. 401–408

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

[36] I. Cox, J. Kilian, T. Leighton, and T. Shamoon, "A secure, robust watermark for multimedia," in *LNCS 1174*, 1996, pp. 317–333.

[37] D. Curran, M. O. Cinneide, N. Hurley, and G. Silvestre, "Dependency in software watermarking," in *First International Conference on Information and Communication Technologies: from Theory to Applications*, 2004, pp. 311–324.

[38] 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*, 2003, pp. 311–324.

[39] R. Davidson and N. Myhrvold, "Computer software protection," *US Patent*, vol. 5,287,407, 1994.

[40] R. Davidson and N. Myhrvold, "Method and system for generating and auditing a signature for a computer program," *US Patent*, vol. 5,559,884, 1996.

[41] H. Delfs and H. Knebl, *Introduction to cryptography, principles and applications.* Springer-Verlag, 2002.

[42] G. Dong, J. Han, J. Lam, J. Pei, K. Wang, and W. Zou, "Mining constrained gradients in large databases," *IEEE Trans. On Knowledge and Data Engineering*, vol. 16, no. 8, pp. 922–938, 2004.

[43] S. Drape, "Obfuscation of abstract data-types," Ph.D. dissertation, The Universite of Oxford, 2004.

[44] R. El-Khalil and A. Keromytis, "Hydan: Embedding secrets in program binaries," in *http://www.andrew.cmu.edu/user/dgao/InfoHiding/binary-stego.pdf*, Aug. 14 2004.

[45] L. Ertaul and S. Venkatesh, "Novel obfuscation algorithms for software security," in *2005 International Conference on Software Engineering Research and Practice, SERP'05*, june 2005, pp. 209–215.

[46] L. Ertaul and S. Venkatesh, "Jhide - a tool kit for code obfuscation," in *8th IASTED International Conference on Software Engineering and Applications (SEA 2004)*, Nov. 2004, pp. 133–138.

[47] O. Esparza, M. Fernandez, M. Soriano, J. Munoz, and J. Forne, "Mobile agent watermarking and fingerprinting tracing malicious hosts," in *LNCS 2736*, 2003, pp. 927–936.

[48] K. Fukushima and K. Sakurai, "A software fingerprinting scheme for java using classfiles obfuscation," in *LNCS 2908*, 2003, pp. 303–316.

[49] J. Ge and S. C. A. Tyagi, "Control flow based obfuscation," in *DRM'05*. ACM, Nov. 2005, pp. 83–92.

[50] D. Gollmann, *Computer Security*. New York: Willey, 1999.

[51] S. Goldwasser and Y. Kalai, "On the Impossibility of Obfuscation with Auxiliary Input," in *FOCS'05*, 2005, pp. 553-562.

[52] D. Grover, *The Protection of Computer Software - Its Technology and Applications*, 2nd ed. Cambridge University Press, 1997.

[53] G. Hachez, "A comparative study of software protection tools suited for e-commerce with contributions to software watermarking and smart cards," Ph.D. dissertation, Universite Catholique de Louvain, Mar 2003.

[54] M. Hall and G. Holmes, "Benchmarking attribute selection techniques for discrete class data mining," *IEEE Trans. On Knowledge and Data Engineering*, vol. 15, no. 6, pp. 1437–1447, 2003.

[55] Y. He, "Tamperproofing a software watermark by encoding constants," Master's thesis, University of Auckland, Mar 2002.

[56] D. He and Q. Sun, "A RST resilient object-based video watermarking scheme," in *ICIP 2004, vol. 2, Singapore, 24–27 october 2004*, pp. 737–740.

[57] E. Hughes, A. Kazura, and A. Rosenthal, "Policy-based information sharing with semantics," in *ISI 2004*, ser. LNCS, vol. 3073, June 2004, pp. 508–509.

[58] K. Ivanov and V. Zakharov, "Program obfuscation as obstruction of program static analysis," in *Volume 6. ISPRAN 2005. Russian Academy of Sciences Technical Report Series*, 2005, pp. 137–156.

[59] M. Jakobsson and M. Reiter, "Discouraging software piracy using software aging," in *Proc. ACM Workshop Security and Privacy in Digital Rights Management*. ACM Press, 2001, pp. 1–12.

[60] R. Jensen and Q. Shen, "Semantics-preserving dimensionality reduction: Rough and fuzzy-rough-based approaches," *IEEE Trans. On Knowledge and Data Engineerin*, vol. 16, no. 12, pp. 1457–1471, 2004.

[61] S. R. Kirk and S. Jenkins, "Information theory-based software metrics and obfuscation," *The Journal of Systems and Software*, vol. 72, pp. 179–186, 2004.

[62] D. Knuth, *The art of computer programming, Vol. 2, Seminumerical algorithms*, 3rd ed. Addison-Wesley, 1997.

[63] C. Kruegel, W. Robertson, F. Valeur, and G. Vigna, "Static disassembly of obfuscated binaries," in *Proceedings of USENIX Security*, Aug 2004, pp. 255–270. [Online]. Available: citeseer.ist.psu.edu/kruegel04static.html

[64] A. Lakhotia, E. U. Kumar, and M. Venable, "A method for detecting obfuscated calls in malicious binaries," *IEEE Transactions on Software Engineering*, vol. 13, no. 11, pp. 955–968, 2005.

[65] B. Lampson, "Computer security in the real world," *Computer*, vol. 37, no. 6, pp. 37–46, 2004.

[66] A. Lang and J. Dittmann, "Transparency and Complexity Benchmarking of Audio Watermarking Algorithms Issues" in *the 8th ACM Multimedia and Security Workshop, 26-27, September, 2006, Geneva, Switzerland*, pp. 190–201.

[67] J. R. Larus, "Whole program paths," in *ACM SIGPLAN Conference on Programming Language Design and Implementation(PLDI 99)*, 1999. [Online]. Available: http://citeseer.ist.psu.edu/stern00robust.html

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

[69] T. Y. Lin, "Granular computing - structures, representations, and applications," in *LNAI*, vol. 2639, 2003, pp. 16–24.

[70] T. Y. Lin, "Granular computing on binary relations-analysis of conflict and chinese wall security policy," in *Rough Sets and Current Trends in Computing*, ser. LNAI, vol. 2475, 2002, pp. 296–299.

[71] T. Y. Lin, "Placing the chinese walls on the boundary of conflicts," in *Proc. 26th Annual International Computer Software and Applications Conference*, 2002, pp. 966–971.

[72] T. Y. Lin, "Chinese walls security model and conflict analysis," in *Proc. 24th Annual International Computer Software and Applications Conference*, 2000, pp. 122–127.

[73] T. Y. Lin and Q. Liu, "Rough approximate operators: axiomatic rough set theory," in *Rough Sets, Fuzzy Sets and Knowledge Discovery*, W. Ziarko, Ed. Springer, 1994, pp. 256–260.

[74] T. Y. Lin, "Chinese walls security policy - an aggresive model," in *Proc. 5th Annual International Computer Software and Applications Conference*, 1989, pp. 282–289.

[75] D. Low, "Protecting java code via code obfuscation," Master's thesis, University of Auckland, 1998.

[76] B. Lynn, M. Prabhakaran, and A. Sahai, "Positive results and techniques for obfuscation," in *EUROCRYPT 2004*, ser. LNCS, vol. 3027, 2004, pp. 20–39.

[77] A. Main and P. van Oorschot, "Software protection and application security: Understanding the battleground," 2004. [Online]. Available: www.scs.carleton.ca/ paulv/papers/softprot8a.ps

[78] A. Majumdar and C. Thomborson, "Manufacturing opaque predicates in distributed systems for code obfuscation," in *Twenty-Ninth Australasian Computer*

Science Conference, V. Estivill-Castro and G. Dobbie, Eds., vol. 48. CRPIT, ACM Digital Library, pp. 187–196.

[79] A. Majumdar and C. Thomborson, "On the use of opaque predicates in mobile agent code obfuscation," in *ISI 2005*, ser. LNCS, vol. 3495, May 2005, pp. 648–649.

[80] B. Aleman-Meza, P. B. M. Eavenson, D. Palaniswami, and A. Sheth, "An ontological approach to the document access problem of insider threat," in *ISI 2005*, ser. LNCS, vol. 3495, 2005, pp. 486–491.

[81] A. Monden, A. Monsifrot, and C. Thomborson, "A framework for obfuscated interpretation," in *Australia Information Security Workshop 2004*, 2004, pp. 7–16.

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

[83] A. Monden, H. Iida, K. ichi Matsumoto, K. Inoue, and K. Torii, "Watermarking java programs," in *International Symposium on Future Software Technology '99*, October 1999, pp. 119–124.

[84] J. Mordeson, "Rough set theory applied to (fuzzy) ideal theory," *Fuzzy Sets and Systems*, vol. 121, pp. 315–324, 2001.

[85] S. Moskowitz and M. Cooperman, "Method for stega-cipher protection of computer code," *US Patent*, vol. 5,745,569, 1994.

[86] P. Moulin and A. Ivanovic, "Game-theoretic analysis of watermark detection," in *ICIP (3)*, 2001, pp. 975–978.

[87] P. Moulin and J. O'Sullivan, "Information–theoretic analysis of watermarking," in *Proc. Int. Conf. on Ac., Sp. and Sig. Proc. (ICASSP)*, 2000.

[88] P. Moulin and J. O'Sullivan, "Information–theoretic analysis of information hiding," *IEEE Transactions on Information Theory*, vol. 49, no. 3, pp. 563–593, 2003.

[89] G. Myles and C. Collberg, "Software watermarking through register allocation: Implementation, analysis, and attacks," in *LNCS 2971*, 2004, pp. 274–293.

[90] G. Myles and C. Collberg, "Software watermarking via opaque predicates: Implementation, analysis, and attacks," in *ICECR-7*, 2004.

[91] G. Myles and C. Collberg, "Detecting software theft via whole program path birthmarks," in *Information Security Conference*, 2004.

[92] J. Nagra and C. Thomborson, "Threading software watermarks," in *IH'04*, 2004.

[93] J. Nagra, C. Thomborson, and C. Collberg, "A functional taxonomy for software watermarking," in *Twenty-Fifth Australasian Computer Science Conference (ACSC2002)*, M. J. Oudshoorn, Ed. Melbourne, Australia: ACS, 2002. [Online]. Available: http://citeseer.nj.nec.com/508809.html

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

[95] A. Narayanan and V. Shmatikov, "Obfuscated databases and group privacy," in *CCS'05*, Alexandria, Virginia, USA, November 7–11 2005, pp. 264–173.

[96] G. Naumovich and N. Memon, "Preventing piracy, reverse engineering, and tampering," *Computer*, vol. 36, no. 7, pp. 64–71, 2003.

[97] J. Nicherson, S. Chow, and H. Johnson, "Tamper resistant software: extending trust into a hostile environment," in *Proceedings of ACM Multimedia '01.* ACM Press, 2001.

[98] P. van Oorschot, "Revisiting software protection," in *ISC 2003*, ser. LNCS, vol. 2851, 2003, pp. 1–13.

[99] S. Pal and P. Mitra, "Case generation using rough sets with fuzzy representation," *IEEE Trans. On Knowledge and Data Engineering*, vol. 16, no. 3, pp. 292–300, 2004.

[100] J. Palsberg, S. Krishnaswamy, K. Minseok, D. Ma, Q. Shao, and Y. Zhang, "Experience with software watermarking," in *Proceedings of the 16th Annual Computer Security Applications Conference, ACSAC '00.* IEEE, 2000, pp. 308–316. [Online]. Available: http://www.cs.purdue.edu/homes/madi/wm/watermarking.ps

[101] Z. Pawlak, "Some remarks on conflict analysis," *European Journal of Operational Research*, vol. 166, pp. 649–654, 2005.

[102] Z. Pawlak, "An inquiry into anatomy of conflicts," *Journal of Information Sciences*, vol. 109, pp. 65–78, 1998.

[103] Z. Pawlak, "Analysis of conflicts," in *Joint Conference of Information Science*, March 1997, pp. 350–352.

[104] Z. Pawlak, *Rough sets: Theoretical aspects of reasoning about data.* Kluwer Academic Publishers, Boston, 1991.

[105] Z. Pawlak, "On conflicts," *International Journal of ManMachine Studies*, vol. 21, pp. 127–134, 1984.

[106] Z. Pawlak, "Rough sets," *Internat. J. Comput. Inform. Sci.*, vol. 11, pp. 341–356, 1982.

[107] L. Polkowski and A. Skowron, Eds., *Rough sets and current trends in computing.* Springer, 1998, vol. 1424.

[108] L. Polkowski and A. Skowron, *Rough sets in knowledge discovery.* Heidelberg: Physica–Verlag, 1998, vol. 1.

[109] L. Polkowski and A. Skowron, *Rough sets in knowledge discovery.* Heidelberg: Physica–Verlag, 1998, vol. 2.

[110] J. A. Pomykala, "Approximation operations in approximation space," *Bull. Pol. Acad. Sci.*, vol. 35, no. 9-10, pp. 653–662, 1987.

[111] M. D. Preda and R. Giacobazzi, "Semantic-based code obfuscation by abstract interpretation," in *Proceedings of the 32th Internarional Colloquium on Automata, Language and Programming*, vol. 3580. Springer Verlag, 2005, pp. 1325–1336.

[112] M. D. Preda and R. Giacobazzi, "Control code obfuscation by abstract interpretation," in *Proceedings of the Third IEEE International Conference on Software Engineering and Formal methods.* IEEE Computer Society, 2005, pp. 1–10.

[113] G. Qu and M. Potkonjak, "Fingerprinting intellectual property using constraint-addition," in *Design Automation Conference '00*, 2000, pp. 587–592.

[114] G. Qu, J. Wong, and M. Potkonjak, "Fair watermarking techniques," in *EEE/ACM Asia and South Pacific Design Automation Conference, '00*, 2000, pp. 55–60.

[115] G. Qu, J. Wong, and M. Potkonjak, "Optimization-intensive watermarking techniques for decision problems," in *Design Automation Conference, '99*, 1999, pp. 33–36.

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

[117] D. Radlauer, "Incident and casualty databases as a tool for understanding low-intensity conflicts," in *ISI 2005*, ser. LNCS, vol. 3495, 2005, pp. 153–170.

[118] J. C. Rabek, R. I. Khazan, S. M. Lewandowski, and R. K. Cunningham, "Detection of injected, dynamically generated, and obfuscated malicious code," in *WORM03*, Washington, DC, USA, October 27 2003, pp. 76–82.

[119] E. Reid and H. Chen, "Mapping the contemporary terrorism research domain: Researchers, publications, and institutions analysis," in *ISI 2005*, ser. LNCS, vol. 3495, May 2005, pp. 648–649.

[120] 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*, vol. TR04–07, March 2004.

[121] Y. Sakabe, "Java obfuscation approaches to construct tamper-resistant object-oriented programs," *IPSJ Digital Courier*, vol. 1, pp. 349–361, 2005.

[122] Y. Sakabe, M. Soshi, and A. Miyaji, "Java obfuscation with a theoretical basis for building secure mobile agents," in *Communications and Multimedia Security*, ser. LNCS, vol. 2828, 2003, pp. 89–103.

[123] P. Samson, "Apparatus and method for serializing and validating copies of computer software," *US Patent*, vol. 5,287,408, 1994.

[124] M. Sipser "Time Complexity, Introduction to the Theory of Computation," 2nd edition, USA: Thomson Course Technology, 2006.

[125] J. S. A. Skowron, "Tolerance approximation spaces," *Fundamenta Informaticae*, vol. 27, pp. 245–253, 1996.

[126] R. Slowinski and D. Vanderpooten, "A generalized definition of rough approximations based on similarity," *IEEE Trans. On Knowledge and Data Engineering*, vol. 12, no. 2, pp. 331–336, 2000.

[127] M. Sosonkin, G. Naumovich, and N. Memon, "Obfuscation of design intent in object-oriented applications," in *DRM'03*. ACM, Oct. 2003, pp. 142–153.

[128] J. Stern, G. Hachez, F. Koeune, and J.-J. Quisquater, "Robust object watermarking: Application to code," in *Information Hiding Workshop '99*, 1999, pp. 368–378. [Online]. Available: http://citeseer.ist.psu.edu/stern00robust.html

[129] M. Stytz and J. Whittaker, "Software protection: security's last stand?" *IEEE Security & Privacy Magazine*, vol. 1, no. 1, pp. 95–98, 2003.

[130] C. Su and J. Hsu, "An extended chi2 algorithm for discretization of real value attributes," *IEEE Trans. On Knowledge and Data Engineering*, vol. 17, no. 3, pp. 437–441, 2005.

[131] H. Tamada, M. Nakamura, A. Monden, and K. Matsumoto, "Design and evaluation of birthmarks for detecting theft of java programs," in *Proc. IASTED International Conference on Software Engineering (IASTED SE2004)*, Feb 2004, pp. 569–575.

[132] S. Thaker, "Software watermarking via assembly code transformations," Master's thesis, San Jose State University, 2004.

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

[134] T. Toyofuku, T. Tabata, and K. Sakurai, "Program obfuscation scheme using random number to complicate control flow," in *EUC Workshops 2005*, ser. LNCS, vol. 2823, 2005, pp. 916–925.

[135] E. Tsang, D. Cheng, J. Lee, and D. Yeung, "On the upper approximations of covering generalized rough sets," in *Proc. 3rd International Conf. Machine Learning and Cybernetics*, 2004, pp. 4200–4203.

[136] S. K. Udupa, S. K. Debray, and M. Madou, "Deobfuscation reverse engineering obfuscate code," in *Proceedings of the 12th Working Conference on Reverse Engineering(WCRE'05)*. IEEE, 2005.

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

[138] C. Wang, "A security architecture for survivability mechanisms," Ph.D. dissertation, University of Virginia, School of Engineering and Applied Science, October 2000, —www.cs.virginia.edu/ survive/pub/wangthesis.pdf—.

[139] F.-Y. Wang, "Outline of a computational theory for linguistic dynamic systems: Toward computing with words," *International Journal of Intelligent Control and Systems*, vol. 2, no. 2, pp. 211–224, 1998.

[140] C. Wang, J. Hill, J. Knight, and J. Davidson, "Software tamper resistance: Obstructing static analysis of programs," University of Virginia, Tech. Rep. CS-2000-12, Dec. 2000. [Online]. Available: citeseer.nj.nec.com/wang00software.html

[141] H. Wee, "On obfuscating point functions," in *STOC'05*, H. N. Gabow and R. Fagin, Eds. ACM, 2005, pp. 523–532.

[142] R. Wille, "Formal concept analysis as mathematical theory of concepts and concept hierarchies," in *Formal Concept Analysis*, ser. LNCS, B. Ganter and et al, Eds., vol. 3626, 2005, pp. 1–33.

[143] R. Wille, "Restructuring lattice theory: an approach based on hierarchies of concepts," in *Ordered sets*, I.Rival, Ed. Reidel, Dordrecht-Boston, 1982, pp. 445–470.

[144] G. Wroblewski, "A general method of program code obfuscation," Ph.D. dissertation, Wroclaw University, 2002.

[145] Z. Xia, Y. Jiang, Y. Zhong, , and S. Zhang, "A novel policy and information flow security model for active network," in *ISI 2004, LNCS*, vol. 3073, June 2004, pp. 42–55.

[146] Y. Yao, "A partition model of granular computing," *LNCS*, vol. 3100, pp. 232–253, August 2004.

[147] Y. Y. Yao, "Granular computing: basic issues and possible solutions," in *Proceedings of the 5th Joint Conference on Information Sciences*, vol. 1, 2000, pp. 186–189.

[148] L. Zadeh, "Fuzzy logic = computing with words," *IEEE Transactions on Fuzzy Systems*, vol. 4, pp. 103–111, 1996.

[149] L. Zadeh, "The concept of a linguistic variable and its application to approximate reasoning – I," *Information Sciences*, vol. 8, pp. 199–249, 1975.

[150] L. Zadeh, "The concept of a linguistic variable and its application to approximate reasoning – II," *Information Sciences*, vol. 8, pp. 301–357, 1975.

[151] L. Zadeh, "The concept of a linguistic variable and its application to approximate reasoning – III," *Information Sciences*, vol. 9, pp. 43–80, 1975.

[152] L. Zadeh, "Fuzzy sets," *Information and Control*, vol. 8, pp. 338–353, 1965.

[153] W. Zakowski, "Approximations in the space (u, π)," *Demonstratio Mathematica*, vol. 16, pp. 761–769, 1983.

[154] B. Zhang and L. Zhang, *Theory and Allication of Problem Solving*. Elsevier Science Publishers B. V., Noth-Holland, 1992.

[155] L. Zhang and B. Zhang, "The quotient sapce theory of problem solving," in *RSFDGrC 2003*, ser. LNCS, vol. 2639, 2003, pp. 11–15.

[156] L. Zhang, Y. Yang, , X. Niu, and S. Niu, "A survey on software watermarking," *Journal of Software*, vol. 14, no. 2, pp. 268–277, 2003.

[157] N. Zhong, Y. Yao, and M. Ohshima, "Peculiarity oriented multidatabase mining," *IEEE Trans. On Knowledge and Data Engineering*, vol. 15, no. 4, pp. 952–960, 2003.

[158] W. Zhu, "Generalized Rough Sets Based on Relations," to appear in *Information Sciences*, 2007.

[159] W. Zhu and F. Y. Wang, "On Three Types of Covering Based Rough Sets," to appear in *IEEE TKDE*, Vol. 19, No. 8, 2007, pp. 1131–1144.

[160] W. Zhu and F. Y. Wang, "Properties of the Third Type of Covering-Based Rough Sets, " in *ICMLC 2007*, Hong Kong, 19–22 August, 2007.

[161] W. Zhu and F. Y. Wang, "Topological properties in Covering-Based Rough Sets," to appear in *FSKD'07*, Haikou , China , 24–27 August, 2007.

[162] W. Zhu, "Basic Concepts in Covering-Based Rough Sets," to appear in *ICNC'07*, Haikou , China , 24–27 August, 2007.

[163] W. Zhu, "A Class of Fuzzy Rough Sets Based on Coverings," to appear in *FSKD'07*, Haikou , China , 24–27 August, 2007.

[164] W. Zhu and F. Y. Wang, Covering-Based Granular Computing," to appear as a book chapter in "Granular Computing: Past, Present, and Future" to be published by Science Press in China, 2007.

[165] B. Zhang, W. Zhu and Z. Xue, "Mining Privilege Escalation Paths For Network Vulnerability Analysis," to appear in *ICNC-FSKD'07*, Haikou , China , 24–27 August, 2007.

[166] W. Zhu, "Informed Recognition in Software Watermarking, in *PAISI 2007*, Chengdu, China, April 11-12, 2007, LNCS 4430, pp. 257C261.

[167] W. Zhu, "Topological approaches to covering rough sets," *Information Sciences*, Vol. 177, No. 6, 2006, pp. 1499–1508.

[168] W. Zhu, "Properties of the Fourth Type of Covering-Based Rough Sets," in *HIS-NCEI'06*, 13–15 December, 2006, New Zealand.

[169] W. Zhu and F. Y. Wang, "Properties of the First Type of Covering-Based Rough Sets," in *ICDM Workshop 2006*, 18–22 December, 2006, Hong Kong.

[170] W. Zhu, "Properties of the Second Type of Covering-Based Rough Sets," in *GrC&BI'06 Workshop 2006*, 18–22 December, 2006, Hong Kong.

[171] W. Zhu and C. Thomborson, "Recognition in Software Watermarking" in *the 1st ACM Workshop on Content Protection and Security, October 27th, 2006, Santa Barbara, CA, USA*.

[172] W. Zhu and C. Thomborson, "Extraction in Software Watermarking" in *the 8th ACM Multimedia and Security Workshop, 26-27, September, 2006, Geneva, Switzerland.*

[173] 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*, 2006, pp. 770–773.

[174] W. Zhu, C. Thomborson, and F.-Y. Wang, "Application of homomorphic function to software obfuscation," in *WISI 2006*, ser. LNCS, vol. 3917, April 2006, pp. 152–153.

[175] W. Zhu and F.-Y. Wang, "Relationships among three types of covering rough sets," in *IEEE GrC 2006*, May 2006, pp. 43–48.

[176] W. Zhu and F.-Y. Wang, "A new type of covering rough sets," in *to appear IEEE IS 2006, 4-6 September 2006*, Sept 2006.

[177] W. Zhu and F.-Y. Wang, "Axiomatic systems of generalized rough sets," in *RSKT 2006*, ser. LNAI, vol. 4062, 2006, pp. 216–221.

[178] W. Zhu and F.-Y. Wang, "Binary relation based rough set," *IEEE FSKD 2006*, 2006, pp. 276–285.

[179] W. Zhu and C. Thomborson, "A Provable Scheme for Homomorphic Obfuscationin in Software Security," in *CNIS'05*, 2005, pp. 208–212.

[180] W. Zhu and C. Thomborson, "Algorithms to watermark software through register allocation," in *DRMTICS 2005*, ser. LNCS, vol. 3919, October 2005, pp. 180–191.

[181] W. Zhu, C. Thomborson, and F.-Y. Wang, "A survey of software watermarking," in *IEEE ISI 2005*, ser. LNCS, vol. 3495, May 2005, pp. 454–458.

[182] W. Zhu and F.-Y. Wang, "Reduction and axiomization of covering generalized rough sets," *Information Sciences*, vol. 152, pp. 217–230, 2003.

[183] F. Zhu, "On covering generalized rough sets," Master's thesis, The Universite of Arizona, Tucson, Arizona, USA, May 2002.

[184] F. Zhu and F.-Y. Wang, "Some results on covering generalized rough sets," *Pattern Recognition and Artificial Intelligence*, vol. 15, no. 1, pp. 6–13, 2002.

[185] F. Zhu and H.-C. He, "The axiomization of the rough set," *Chinese Journal of Computers*, vol. 23, no. 3, pp. 330–333, March 2000.

[186] F. Zhu and H.-C. He, "Logical properties of rough sets," in *Proc. of The Fourth International Conference on High Performance Computing in the Asia-Pacific Region*, vol. 2. IEEE Press, 2000, pp. 670–671.

[187] X. Zhuang, T. Zhang, H.-H. S. Lee, and S. Pande, "Hardware assisted control flow obfuscation for embedded processors," in *CASES'04*, Washington, DC, USA, September 22-25 2004, pp. 292–302.

Appendix A

Publications

Publications During My PhD Study at the University of Auckland

1. William Zhu, Generalized Rough Sets Based on Relations, to appear in Information Sciences, 2007.

2. William Zhu and Fei-Yue Wang, On Three Types of Covering Based Rough Sets, IEEE Transactions on Knowledge and Data Engineering, Vol. 19, No. 8, 2007, pp. 1131–1144.

3. William Zhu, Fei-Yue Wang, Properties of the Third Type of Covering-Based Rough Sets, to appear in ICMLC 2007, Hong Kong, 19–22 August, 2007.

4. William Zhu, Fei-Yue Wang, Topological properties in Covering-Based Rough Sets, to appear in FSKD'07, Haikou , China , 24–27 August, 2007.

5. William Zhu, Basic Concepts in Covering-Based Rough Sets, to appear in ICNC'07, Haikou , China , 24–27 August, 2007.

6. William Zhu, A Class of Fuzzy Rough Sets Based on Coverings, to appear in FSKD'07, Haikou , China , 24–27 August, 2007.

7. William Zhu, Fei-Yue Wang, Covering-Based Granular Computing, to appear as a book chapter in "Granular Computing: Past, Present, and Future" to be published by Science Press in China ,2007.

8. Baowen Zhang, William Zhu, Zhi Xue, Mining Privilege Escalation Paths For Network Vulnerability Analysis, to appear in ICNC-FSKD'07, Haikou , China , 24–27 August, 2007.

9. William Zhu, Informed Recognition in Software Watermarking, in PAISI 2007, Chengdu, China, April 11-12, 2007, LNCS 4430, pp. 257C261.

10. William Zhu, Properties of the Fourth Type of Covering-Based Rough Sets, in HIS-NCEI'06, 13–15 December, 2006, New Zealand.

11. William Zhu, Fei-Yue Wang, Properties of the First Type of Covering-Based Rough Sets, in ICDM Workshop 2006, 18–22 December, 2006, Hong Kong.

12. William Zhu, Properties of the Second Type of Covering-Based Rough Sets, in GrC&BI'06 Workshop 2006, 18–22 December, 2006, Hong Kong.

13. Han Zhang, Gerald Weber, William Zhu, Clark Thomborson, B2B E-commerce Security Modeling – Case Study, in International Conference on Computational Intelligence and Security (CIS2006), 3-6 November, 2006, Guangzhou, China.

14. William Zhu, Clark Thomborson, Recognition in Software Watermarking, in 1st ACM Workshop on Content Protection and Security, October 27th, 2006, Santa Barbara, CA, USA.

15. William Zhu, Clark Thomborson, Extraction in Software Watermarking, in ACM

Multimedia and Security Workshop 2006, 26-27 September, 2006, Geneva, Switzerland, pp. 175–181.

16. William Zhu, Topological Approaches to Covering Rough Sets, Information Sciences, Vol. 177, 2006, 1499–1508.

17. William Zhu, Dualities in Covering Rough Operations, in IFTGrCRSP 2006, 20-22 July, 2006, Nanchang, China, pp. 59–63.

18. William Zhu and Fei-Yue Wang, Binary Relation Based Rough Set, in IEEE FSKD 2006, 24-28 September, 2006, Xi'an, China, LNAI 4223, pp. 276-285.

19. William Zhu and Fei-Yue Wang, A New Type of Covering Rough Sets, in IEEE IS 2006, 4-6 September, 2006, London, UK, pp. 444–449.

20. William Zhu and Fei-Yue Wang, Axiomatic Systems of Generalized Rough Sets, in RSKT 2006, 24-26 July, 2006, Chongqing, China, LNAI 4062, pp. 216-221.

21. William Zhu and Fei-Yue Wang, Covering Based Granular Computing for Conflict Analysis, in IEEE ISI 2006, 22-24 May, 2006, San Diego, CA, USA, LNCS 3975, pp. 566–571.

22. William Zhu, Clark Thomborson, and Fei-Yue Wang, Obfuscate Arrays by Homomorphic Functions, in IEEE GrC 2006, 10-12 May, 2006, Atlanta, GA, USA, pp. 770–773.

23. William Zhu and Fei-Yue Wang, Relationships among Three Types of Covering Rough Sets, in IEEE GrC 2006, 10-12 May, 2006, Atlanta, GA, USA, pp. 43–48.

24. William Zhu, Clark Thomborson, and Fei-Yue Wang, Application of Homomorphic Function to Software Obfuscation, in WISI 2006, April 9, 2006, Singpore, LNCS 3917, pp. 152–153.

25. William Zhu and Clark Thomborson, A Provable Scheme for Homomorphic Obfuscationin in Software Security, in CNIS 2006, 14-16 November, 2005, Phoenix, AZ, USA, pp. 208–212.

26. William Zhu and Clark Thomborson, Algorithms to Watermark Software through Register Allocation, in DRMTICS 2005, Oct 30 - Nov. 1, 2005, Sydney, Australia, LNCS 3919, pp. 180–191.

27. William Zhu, Clark Thomborson, and Fei-Yue Wang, A Survey of Software Watermarking, in IEEE ISI 2005, Atlanta, GA, USA, LNCS 3495, pp. 454–458.

28. William Zhu and Clark Thomborson, On the QP Algorithm in Software Watermarking, in IEEE ISI 2005, Atlanta, GA, USA, LNCS 3495, pp. 646–647.

Previous Publications

1. William Zhu and Fei-Yue Wang, Reduction and Axiomization of Covering Generalized Rough Sets, Information Sciences, 152(2003), pp. 217–230.

2. Feng Zhu and Fei-Yue Wang, Some Results on Covering Generalized Rough Sets, Pattern Recognition & Artificial Intelligence(in Chinese), Vol.15, No.1 (2002) 6-13.

3. Feng Zhu and Huacan He, Logical Properties of Rough Sets, Proceedings of TheFourth International Conference on High Performance Computing in the Asia-Pacific Region, Los Alamitos, CA: IEEE Press, Vol.2(2000) 670-671.

4. Feng Zhu and Huacan He, The axiomatization of the rough set, Chinese Journal of Computers (in Chinese), Vol.23, No.3 (2000) 330-333.

5. Feng Zhu and Huacan He, Characteristics of Rough Sets, Journal of Northwestern Polytechnical University (in Chinese), Vol. 19, No.3 (2001) 422-425.

6. Feng Zhu and Huacan He, The Logical Properties of Lower and Upper Approximation Operations in Rough Sets, Computer Science (in Chinese), Vol. 27, No. 11 (2000) 79-81.

7. Feng Zhu and Huacan He and etc., The Rough Elements in the Rough Set: Their Structures and Generalizations, Computer Science (in Chinese), Vol. 27, No. 6 (2000) 38-39,34.

8. Feng Zhu and Huacan He and etc., Properties of Rough Sets in Topological Boolean Algebra, Microelectronics & Computer(Special Issue) (in Chinese), 1999.

9. Yuanquan Zhou and Huacan He and Feng Zhu, Association rules based on Generalized Logic, Microelectronics & Computer(Special Issue) (in Chinese), 1999.

10. William Zhu, Consistency Relations in the Theory of Rough Sets, Journal of Huaqiao University (in Chinese), Vol.22, No.2 (2001) 217-220.

Appendix B

Academic services

Member of the Programme Committee:

1. GrC 2007, 2-4 November, Silicon Valley, USA
2. ICMLC2007,Hong Kong, 19-22 August, 2007.
3. ICNC'07, Haikou, China, 24–27 August, 2007.
4. ISI 2007, Hyatt Hotel, New Brunswick, NJ, 23 - 24 May, 2007.
5. PAISI 2007, Chengdu, China, April 11-12, 2007.
6. PRICAI 2006, Guilin, China, August 7-11, 2006.
7. RSKT 2006 in July 24-26, 2006, Chongqing, China.
8. IEEE ISI 2006, San Diego, USA, 22-24 May, 2006.
9. WISI 2006 in Singapore on 9 April, 2006.

Reviewer:

1. IEEE ISI 2005, Atlanta, GA, USA 19-20 May, 2005.

2. CogSci 2006, Vancouver, BC, Canada July 26-29, 2006.

3. ICCS 2006, Vancouver, BC, Canada July 26-29, 2006.

4. International Journal of Approximate Reasoning.

5. IEEE Transaction on Knowledge and Data Engineering.

6. Journal of Systems and Software.

7. Soft Computing.

8. Information Sciences.

Session Chair:

1. The Session on Rough Computing in IEEE GrC 2006, Atlanta, GA, USA 10-12 May, 2006.

2. IFTGrCRSP2006, 20 July, 2006, Nanchang, China.

3. The Session on Rough Computing in RKST 2006, Chongqing, China, 25 July, 2006.

Panellist:

1. IFTGrCRSP2006, 20 July, 2006, Nanchang, China.

Invited Talk:

1. College of Information Science, Beijing Language and Culture University, 23 May, 2007.

2. College of Computer Information Engineering, Jiangxi Normal University, Nanchang, China, 25 April - 17 May, 2007.

3. Department of Computer Science, Nanchang Institute of Technology, Nanchang, China, 16 May, 2007.

4. UFIDA School of Software, Jiangxi University of Finance and Economics, Nanchang, China, 12 May, 2007.

5. Hunan University, Changsha, China, 4 January, 2007.

6. The First Summer School for Postgraduates of Jiangxi Province, July 19, 2006.

Visiting Scholar:

1. Chern Institute of Mathematics, Nankai University, Tianjin, China, 18–30 May, 2007.

2. Chern Institute of Mathematics, Nankai University, Tianjin, China, 1–31 December, 2006.

3. Chern Institute of Mathematics, Nankai University, Tianjin, China, 20–25 September, 2006.