Robust Digital Watermarking of Multimedia Objects

by

Gaurav Gupta,

Dissertation

Presented to

Department of Computing,

Macquarie University

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Macquarie University

August 2008

The Dissertation Committee for Gaurav Gupta
certifies that this is the approved version of the following dissertation:

Robust Digital Watermarking of Multimedia Objects

Committee:

Professor Josef Pieprzyk, Supervisor

Dr Hua Xiong Wang, Co-Supervisor

Contents

List of Tables **v**

List of Figures **vii**

List of Algorithms **ix**

Acknowledgments **xiii**

Abstract **xv**

Statement of Candidate **xix**

List of Publications **xxi**

Notations Used **xxiii**

Chapter 1 Introduction **1**

1.1 Digital Watermarking 1
1.2 Digital Fingerprinting 3
1.3 Motivation 3
1.4 Contributions 4

Chapter 2 Background **7**

2.1 Fundamental Mathematics 7
2.2 Statistics 9
2.3 Cryptography 11
2.4 Hash Functions 15
2.5 Natural Language Documents 16
2.6 Software 21

2.7 Databases 25

Chapter 3 Overview of watermarking **27**
3.1 Approaches to Watermarking 30
3.2 Text and Natural Language Watermarking 32
3.3 Software Watermarking 39
3.4 Database Watermarking 56
3.5 Conclusion 73

Chapter 4 Natural Language Watermarking **74**
4.1 Current Scenario 75
4.2 Outline of Proposed Scheme 76
4.3 Proposed Scheme 78
4.4 Analysis 86
4.5 Experimental Results 88
4.6 Conclusion 89

Chapter 5 Software Watermarking **91**
5.1 Description of Myles and Jun Watermarking Scheme 94
5.2 Proposed Attack 97
5.3 Implementation Details and Results 100
5.4 Surviving the Debugging Attack 103
5.5 Analysis 106
5.6 Conclusion 106

Chapter 6 Semi-blind and Reversible Database Watermarking **109**
6.1 Introduction 109
6.2 Related Work and Agrawal-Kiernan Scheme 110
6.3 Analysis of Agrawal-Kiernan Watermarking Scheme 113
6.4 Modified Algorithms 116
6.5 Analysis 117
6.6 Conclusion 124

Chapter 7 Blind and Reversible Database Watermarking **125**
7.1 Introduction 125
7.2 Model of Adversary 129
7.3 Proposed Scheme 130

7.4 Experimental Results 131
7.5 Analysis 135
7.6 Conclusion 138

Chapter 8 Conclusion and future research **139**
8.1 Thesis Summary 139
8.2 Future Research Directions 142

Bibliography **144**

Vita **154**

List of Tables

3.1 Comparative study of text watermarking schemes 40
3.2 Comparative study of software watermarking schemes 55
3.3 Meal table 58
3.4 Combination table 59
3.5 Version 1 of combination table 60
3.6 Version 2 of combination table 60
3.7 Original Table 62
3.8 Watermarked with bit 1 62
3.9 Watermarked with bit 0 63
3.10 Foreign exchange rates 68
3.11 Foreign exchange rates (watermarked) 68
3.12 Table with modified primary key 69
3.13 Table with binary representation of numerical values 71
3.14 Watermarked table with binary representation of numerical values . 71
3.15 Owner identification possibilities 72

4.1 Natural language and text watermarking methods 76
4.2 Pseudo-randomization of watermarking sequence 80
4.3 Comparison of empirical results with theoretical values 81
4.4 Illustration of majority voting 86
4.5 Text modification with increasing watermark size 89
4.6 Text amplification with increasing watermark size 89

6.1 Original foreign exchange rates relation 114
6.2 Watermarked foreign exchange rates relation 114
6.3 Probability of success for bit flipping attack 119

6.4 Detecting watermarks in multi-party environment 121

List of Figures

3.1 Bishop's crosier (Australia), 16th century 28
3.2 Watermarks in Australian currency bill 28
3.3 Watermarks in German currency bill 29
3.4 Watermark in Spanish document from 17th century 29
3.5 Magnified view of watermark from Figure 3.4 30
3.6 Inserting intermediate code without effecting output 42
3.7 $61 \times 73 = 3.6^4 + 2.6^3 + 3.6^2 + 4.6^1 + 1.6^0$ in Radix-6 encoding [29] . 45
3.8 Planted Planar Cubic Tree [29] 45
3.9 Watermarks 010 and 111 resulting in the same watermarked graph . 50
3.10 Launching an attack on second-LSB based watermarking 64

4.1 Generating a paragraph permutation using AES 79
4.2 Keys required to get a valid permutation using AES-128 82

5.1 Branch function modifying return addresses 93
5.2 Function set F invoked using secret input parameter key_{AM} 96
5.3 Fingerprint branch function modifies the return address itself 107
5.4 Calling instruction modifies address using key returned by fingerprint branch function 107

6.1 Owner identification 122
6.2 Multiple watermarking scenario - dotted lines denote distortion and solid lines denote watermarking 122

7.1 Effect of changing fraction of tuples marked on detection 134
7.2 Effect of changing percentage of marks that need to be detected to establish watermark presence 134

7.3 Effect of changing attack levels on detection 135

List of Algorithms

1 Euclid's algorithm 8
2 Euclid's extended algorithm 8
3 RSA key generation 13
4 RSA encryption 13
5 RSA decryption 13
6 RSA digital signature generation [68] 15
7 RSA digital signature verification [68] 15
8 Watermark insertion changing inter-word spacing 34
9 Watermarking using collocationally-based synonymy 36
10 Natural language watermarking [14] 40
11 QP watermark insertion [75, 76] 48
12 QP watermark extraction [75, 76] 49
13 QPS watermark insertion[70] 51
14 QPS watermark extraction [70] 52
15 Watermark insertion in numeric set 58
16 Uniform distribution attack 64
17 Watermark insertion [11] 66
18 Watermark detection [11] 66
19 Sentence sequence generation 80
20 Natural language watermark insertion 85
21 Watermark insertion [11] 111
22 Watermark detection [11] 112
23 Reversible and semi-blind watermark insertion 117
24 Reversible and semi-blind watermark detection 118
25 Semi-blind owner identification 119
26 Reversible and blind watermark insertion 132

27 Reversible and blind watermark detection 133
28 Blind owner identification 137

To Gunjan for all her love and support. And my parents and Tina for being the wonderful people they are

Acknowledgments

In our life, we come across many people who inspire and motivate us, who help us become a better person and a better professional. I would like to take this opportunity to thank them for all they have done for me.

Firstly, I thank Josef for his tremendous support, not only for my research, but also for my academic and teaching interests. Thanks to Huaxiong as well for providing excellent guidance in the brief absence of my main supervisor. I appreciate the assistance provided by Daniel, Saurabh, Krystian, Vijaykrishnan, and Simon during various stages of my research. I thank Mohan for introducing me to the interesting topic of digital watermarking during my master's degree and taking the time to supervise me for my master's dissertation. I would also like to thank all my friends who have made a positive difference in my life - Gunjan, Ravi, Maya, Anjali, Jagrat, Colwin, Gautam, Urvi, Mohit, Reema, Meeta, Teju, Eric, Raghu, Radhika, Daniel, Menno, and Tanja. I thank Gunjan's parents, Shekhar and Shobha, for their belief as well.

Thanks to Prof. Banerjee for bringing out the best in me during my undergraduation. He is the best teacher I have ever had and a huge inspiration for me. Very special thanks to Michelle for taking the time to read my thesis and giving her valuable feedback, it is really appreciated.

Most importantly, I thank my parents and my sister for being so kind, loving, and nurturing, despite the brat that I was. They always showed confidence in me and support and appreciated me for what I am.

I also want to acknowledge my late friend, Ashish, one of the nicest guy I have ever met, one who was the best at everything he did (and made us jealous in the process). I know he is in a better place; may his soul rest in peace.

The last part is the trickiest one; I want to acknowledge Gunjan's support during all the seven years that we've been together and four years that we have been married (not that I am keeping a count!), but at the same time thanking her for all she has done makes her indirect contributions towards this thesis look so petty. I would just like to take a moment to appreciate how she appreciated my work, instilled confidence in me and applauded every little success I had in my research as if I had won an Olympic medal. So, far all that, and more, I love you Gunjan.

Abstract

Robust Digital Watermarking of Multimedia Objects

Publication No. _____

Gaurav Gupta

Macquarie University, 2008

Supervisor: Professor Josef Pieprzyk

Digital watermarking has generated significant research and commercial interest in the past decade. The primary factors contributing to this surge are widespread use of the Internet with improved bandwidth and speed, regional copyright loopholes in terms of legislation; and seamless distribution of multimedia content due to peer-to-peer file-sharing applications.

Digital watermarking addresses the issue of establishing ownership over multimedia content through embedding a watermark inside the object. Ideally, this watermark should be detectable and/or extractable, survive attacks such as digital reproduction and content-specific manipulations such as re-sizing in the case of images, and be invisible to the end-user so that the quality of the content is not

degraded significantly. During detection or extraction, the only requirements should be the secret key and the watermarked multimedia object, and not the original unmarked object or the watermark inserted. Watermarking scheme that facilitate this requirement are categorized as blind. In recent times, reversibility of watermark has also become an important criterion. This is due to the fact that reversible watermarking schemes can provided security against secondary watermarking attacks by using backtracking algorithms to identify the rightful owner. A watermarking scheme is said to be reversible if the original unmarked object can be regenerated from the watermarked copy and the secret key.

This research covers three multimedia content types: natural language documents, software, and databases; and discusses the current watermarking scenario, challenges, and our contribution to the field. We have designed and implemented a natural language watermarking scheme that uses the redundancies in natural languages. As a result, it is robust against general attacks against text watermarks. It offers additional strength to the scheme by localizing the attack to the modified section and using error correction codes to detect the watermark. Our first contribution in software watermarking is identification and exploitation of weaknesses in branch-based software watermarking scheme proposed in [71] and the software watermarking algorithm we present is an improvised version of the existing watermarking schemes from [71]. Our scheme survives automated debugging attacks against which the current schemes are vulnerable, and is also secure against other software-specific attacks. We have proposed two database watermarking schemes that are both reversible and therefore resilient against secondary watermarking attacks. The first of these database watermarking schemes is semi-blind and requires the bits modified during the insertion algorithm to detect the watermark. The second scheme is an upgraded version that is blind and therefore does not require anything except a secret key and the watermarked relation. The watermark has a 89% probability of survival even when almost half of the data is manipulated. The

watermarked data in this case is extremely useful from the users' perspective, since query results are preserved (i.e., the watermarked data gives the same results for a query as the unmarked data).

The watermarking models we have proposed provide greater security against sophisticated attacks in different domains while providing sufficient watermark-carrying capacity at the same time. The false-positives are extremely low in all the models, thereby making accidental detection of watermark in a random object almost negligible. Reversibility has been facilitated in the later watermarking algorithms and is a solution to the secondary watermarking attacks. We shall address reversibility as a key issue in our future research, along with robustness, low false-positives and high capacity.

Statement of Candidate

Statement of Candidate

I certify that the work in this thesis entitled "Robust Digital Watermarking of Multimedia Objects" has not previously been submitted for a degree nor has it been submitted as part of requirements for a degree to any other university or institution other than Macquarie University.
I also certify that the thesis is an original piece of research and it has been written by me. Any help and assistance that I have received in my research work and the preparation of the thesis itself have been appropriately acknowledged.
In addition, I certify that all information sources and literature used are indicated in the thesis.

Signature:

Gaurav Gupta - 40478890

Sydney, 08-August-2008

List of Publications

1. Gaurav Gupta, Josef Pieprzyk, and Huaxiong Wang. An Attack-Localizing Watermarking Scheme for Natural Language Documents. In *Proceedings of ACM Symposium on Information, Computer and Communications Security (ASIACCS) 2006*, pages 157 - 165, Taipei, Taiwan, May 2006

2. Gaurav Gupta and Josef Pieprzyk. A Low-Cost Attack on Branch-Based Software Watermarking Scheme. In *Proceedings of Fifth International Workshop on Digital Watermarking (IWDW) 2006*, pages 282-293, Jeju Island, South Korea, November 2006

3. Gaurav Gupta and Josef Pieprzyk. Software Watermarking Resilient to Debugging Attacks. In *Journal of Multimedia*, Volume 2, Number 2, pages 10-16, Academy Publisher, April 2007

4. Gaurav Gupta and Josef Pieprzyk. Reversible and Semi-blind Relational Database Watermarking. In *Proceedings of International Conference on Signal Processing and Multimedia Applications (SIGMAP) 2007*, pages 283-290, Barcelona, Spain, July 2007

5. Gaurav Gupta and Josef Pieprzyk. Reversible and Blind Database Watermarking Using Difference Expansion. In *Proceedings of e-Forensics 2008*, Adelaide, Australia, January 2008

6. Gaurav Gupta and Josef Pieprzyk. Source Code Watermarking Based on

Function Dependency Oriented Sequencing. In *Proceedings of Fourth International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIHMSP) 2008*, Harbin, China, August 2008

Notations Used

1. $\{a_1, \ldots, a_n\}$: set of n elements.
2. $\mathcal{H}(x)$: hash of x.
3. R : relation
4. r : tuple
5. A_i : i^{th} attribute
6. $r.A_i$: i^{th} attribute in tuple r
7. A_i^j : j^{th} LSB of i^{th} attribute where LSB stands for least significant bit
8. $r.A_i^j$: j^{th} LSB of i^{th} attribute in tuple r
9. $r.P$: primary key of tuple r
10. $\|$: concatenation
11. $\mathcal{H}()$: one-way hash function
12. $R \xrightarrow{ins(p)} R_w$: relation R_w is the result of party p inserting its watermark in relation R,
13. $R_w \xrightarrow{det(p)} R$: original relation R is restored by the party p from the watermarked relation R_w
14. $|x|$: size of x in bits

15. $abs(x)$: absolute value of x

16. $\lfloor x \rfloor$: greatest integer smaller than x (floor function)

17. $\lceil x \rceil$: smallest integer greater than x (ceiling function)

18. *Distance* for attribute $r.A_i$: $\delta_{r.A_i} = min_{\tilde{r} \neq r}\{abs(r.A_i - \tilde{r}.A_i)\}$

Chapter 1

Introduction

1.1 Digital Watermarking

Traditionally, copyright logos and company seals have been used to prove ownership and authenticity, respectively. Such measures are sufficiently secure provided that the documents can only be transferred and copied physically. However, this is rarely the case in the digital era. With the increasing popularity of peer-to-peer (P2P) software and fast Internet connections, multimedia objects are manipulated, modified, and transferred illegally over the Internet. Hence, it is no longer sufficient for the objects to merely contain a visible logo. Digital watermarking involves placing a copyright mark in a multimedia object so that the owner can establish his/her right on the multimedia content. Numerous research projects including those described in [99, 62, 63, 60, 73, 15, 36, 58, 64, 84, 104, 61, 106, 105, 107, 97, 108] provide formal definitions and requirements of watermarking and propose watermarking models catering for these conditions. The essential characteristics of a digital watermarking scheme are,

1. *detectability/ extractability*: The watermark should establish the owner's identity during the detection/extraction phase otherwise the watermark is useless. In extractable watermarking schemes, the output of the owner identification algorithm is a bitstring corresponding to the owner's identity, while in detectable watermarking schemes, the output is $true/false$ for a particular potential owner.

2. *robustness*: This refers to the capability of the watermarking scheme to survive

deliberate (for example, modifying, adding, deleting part of the data) and unintentional attacks (for example, digital reproduction and photocopying). Thus, the watermark should be detectable even in an object modified by the attacker.

3. *imperceptibility*: The embedded watermark should not degrade the quality of the multimedia content significantly and should not interfere with the user's interaction with the multimedia object. For example, in images, following are the two drawbacks of visible/ perceptible watermarks.

   (a) the watermark deteriorates the image quality.

   (b) attackers can crop the watermark area to erase the watermark.

4. *low false positive*: False positive is the situation where a company's watermark is detected in an object which was not actually marked by the company. In simpler terms, it means *accidently* detecting a watermark in an object. The probability of such an event should be as small as possible. False positive rates of 10^{-9} or lower are generally considered acceptable. This implies that the watermark will be *accidently* detected in one out of one billion random objects. For example, [11] has a false positive rate of around 10^{-10}.

5. *randomness*: The watermark should be pseudo-randomly distributed across the multimedia object and this distribution should be based on a secret key owned by the owner so that the attacker cannot locate the watermark position.

6. *blindness*: The only element required to detect the watermark in a supposedly watermarked object is a secret key K that is independent of the multimedia object and known to the owner of the object. Especially, the original object should not be required to detect the watermark.

7. *limited distortion*: This relates to the extent to which quality of the multimedia object is degraded after watermarking. The distortion should be kept within a *tolerable range*. What exactly is the tolerable range is determined by the specific situation. Some fields such as military organizations would like to maintain data quality to a certain extent even after watermarking their data while television agencies might be more liberal towards the distortion introduced by watermarking so that the viewer gets low quality data and has to pay for the high quality version. An example of tolerance criteria in the

case of relational databases is query preservation. relations in a database are watermarked, the attributes should not be modified beyond a certain range otherwise the database is likely to return wrong results to queries. Limiting the distortion directly results in satisfying the *imperceptibility* condition discussed above.

8. *reversibility*: We should be able to re-generate the original document from the watermarked document. This makes online content distribution simpler since the clients can download trial versions and generate full versions without having to re-download the entire object again. Majority of the existing watermarking schemes do not address this requirement, but we demonstrate the advantages of reversibility including its application in fighting secondary watermarking attacks (where an attacker watermarks an already watermarked relation and claims ownership of the content).

1.2 Digital Fingerprinting

While watermarking establishes the publisher's identity, *fingerprinting* identifies the buyer to whom the multimedia object is sold. If the buyer distributes the object illegally, then its identity should be extractable from the multimedia content. A unique mark has to be inserted in copies sold to different users. These marks are called digital *fingerprints*. An important requirement of fingerprinting schemes is to be resistant against *collusion attacks*. Let the multimedia object be distributed to n users. A collusion attack is when c out of n users collude and either erase the fingerprint mark completely or modify the object such that it points to one of the $n - c$ users who are not a part of the conspiracy/ collusion. Boneh and Shaw have proposed collusion secure codes in [16].

1.3 Motivation

In recent years, telecommunication technology has improved at a rapid pace, residential and corporate Internet connections have become extremely fast with massive download limits. Freely and cheaply available Internet has steered many individuals towards illegal file sharing and distribution. Median Internet speeds are estimated to be 61 megabits per second (Mbps) in Japan, 45Mbps in South Korea, 17Mbps in France and 7Mbps in Canada [21]. Also, video compression schemes such as DivX

enable a typical-length Hollywood movie to be packed in approximately 700MB. A file of this size can be downloaded in as little as 90 seconds with such fast Internet connections. Digital watermarking and fingerprinting have extensive commercial applications in the software distribution, online music industry, digital content distribution, and in countering the threat of illegal file sharing via peer-to-peer networks. Some of the typical scenarios in which digital watermarking would be useful are as follows.

- Companies want to make their text reports available to the public while maintaining their ownership over the document.
- Organizations develop applications and want to establish their ownership over whole or part of the software.
- GPS companies implementing state-of-the-art real-time software systems for generating maps want to protect the software from being duplicated by rivals.

To understand watermarking, various models were studied, analyzed and compared. The weaknesses in the current watermarking techniques were identified and addressed. These include vulnerability to debugging attacks in software watermarking, irreversibility and susceptibility to secondary watermarking in database watermarking. We also made significant progress in understanding the multimedia-specific watermarking techniques. This is critical given that watermarks are embedded in different multimedia objects by exploiting the inherent redundancies which vary from one multimedia object to another. For example, software objects have instruction-based redundancy since it is possible to convert one set of instructions to another set without effecting the output of the program. On the other hand, natural language documents have word or sentence based redundancy and so we can replace a word by one of its synonym without causing significant loss of document quality. The development of new watermarking techniques are based on tools from Cryptology (such as hashing algorithms or pseudo-random generators) and Coding Theory (such as majority voting and traitor tracing codes).

1.4 Contributions

Our major contributions to the field are listed below.

1. *Natural Language Watermarking [49]*: We have designed semantics-based natural language watermarking scheme that can survive any format-based or synonymy-based attacks. The watermarking scheme embeds a watermark that identifies the owner (publisher) and a fingerprint that identifies the user (buyer). The scheme uses Boneh-codes [16] to construct the fingerprint codes and is resistant against collusion attacks in which multiple attackers conspire to try and destroy the watermark. It also localizes the attack to the section of the document modified by the attacker. The other sections in the document are not effected by the manipulation. Apart from the trivial format-based attacks, the watermark can also survive text addition, deletion, modification (through transformations) and paragraph shuffling attacks.

2. *Software Watermarking [43, 46, 48]*: We have proposed a simple yet effective scheme that exploits the condition C and C++ programming languages place on the source codes to embed watermark in a source file. A program containing n functions can embed an $n - bit$ watermark inside it.

   We have identified that the branch-based watermarking scheme [71], which is one of the stronger of the developed software watermarking techniques suffers from trivial debugging attacks where the attacker can launch an mostly automated attack by using debugger capabilities such as breakpoints and stack tracing. We have shown such an attack on this scheme and presented a modified scheme [46] that survives this category of attack and has no adverse effect on other counts of security or watermark carrying capacity.

3. *Database Watermarking [44, 47]*: The major weaknesses that were identified in watermarking schemes published in [86, 42, 61, 62, 101, 102, 103] (most of them based on [11, 12]) were as follows,

   (a) lack of query preservation, where answers to queries on watermarked databases are different from answers to the same queries on the original database, thereby causing a serious usability problem, and,

   (b) susceptibility to secondary watermarking, where it is impossible to detect the correct owner when the attacker embeds his/her watermark on top of owner's watermark

   We have proposed two reversible watermarking schemes [44, 47] that regenerate the original database relation from the watermarked relation and showed

how this can help us identify the correct owner in case multiple parties watermark it sequentially. The schemes also addresses query-preservation, thus providing better usability from the user's perspective.

Chapter 2

Background

2.1 Fundamental Mathematics

In this section, we provide an overview of fundamental mathematical theorems that lie at the core of, and provide the framework for cryptographic and security techniques. Some of the elements we discuss are Euclid's algorithm, prime numbers and Chinese remainder theorem. Below are fundamental definitions that frequently appear throughout this thesis.

Definition 2.1.1 The set of integers $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$ is denoted by $\mathbb{Z}$.

Definition 2.1.2 Let a, b be two integers. Then a divides b if there exists $c \in \mathbb{Z}$, such that $b = ac$. If a divides b, this is represented by $a \mid b$.

Definition 2.1.3 An integer c is said to be a *common divisor* of integers a, b if $c \mid a, c \mid b$.

Definition 2.1.4 *Greatest Common Divisors: Greatest common divisor, or* gcd*, of two integers a, b such that $a < b$ is the highest integer $g \leq a$ which divides both a and b.*

The application of gcd in modulo arithmetic make them interesting for cryptographers. Euclid's algorithm provided in Algorithm 1 provides us with a method of finding gcd, g of two integers a, b. Euclid's extended algorithm described in Algorithm 2 find integers x, y such that $ax + by = d$.

Input : Integers a, b, such that $a > b$
Output: gcd(a, b)=g, such that $g \mid a, g \mid b$
while $b \neq 0$ **do**
  $temp = a \pmod{b}$;
  $a = b$;
  $b = temp$;
end
Return a;

Algorithm 1: Euclid's algorithm

Input : Integers a, b, such that $a > b$,
Output: gcd(a, b)=g, x, y such that $g \mid a, g \mid b$, $ax + by = d$
if b==*0* **then**
  $d = a, x = 1, y = 0$;
else
  $x_1 = 0, x_2 = 1, y_1 = 1, y_2 = 0$;
  while $b \neq 0$ **do**
    $q = \lfloor (a/b) \rfloor$;
    $r = a - qb$;
    $x = x_2 - qx_1$;
    $y = y_2 - qy_1$;
    $a = b$;
    $b = r$;
    $x_2 = x_1$;
    $x_1 = x$;
    $y_2 = y_1$;
    $y_1 = y$;
  end
  $d = a$;
  $x = x_2$;
  $y = y_2$;
  Return (d, x, y);
end
Return a;

Algorithm 2: Euclid's extended algorithm

Definition 2.1.5 An integer $p \geq 2$ is called *prime* if its only positive divisors are 1 and p. Otherwise, p is said to be *composite*.

Definition 2.1.6 Two integers a, b are called *relatively prime* or *coprime* if $\gcd(a, b)$=1.

2.1.1 Chinese Remainder Theorem (CRT)

Theorem 2.1.1 *If the integers $n_1, n_2, \ldots n_k$ are pairwise relatively prime, then the system of simultaneous congruences,*

$$\begin{aligned} x &\equiv a_1 \pmod{n_1} \\ x &\equiv a_2 \pmod{n_2} \\ &\vdots \\ x &\equiv a_k \pmod{n_k} \end{aligned}$$

has a unique solution modulo $n = \prod_{i=1}^{k} n_i$.

The application of CRT lies in performing calculations in RSA, discussed in Section 2.3.2. The calculations in RSA are made modulo n where n is a product of two large primes p, q, that are either 1024, 2048, or 4096-bit integers. Using CRT, the time requirement on these operations is greatly reduced.

2.2 Statistics

2.2.1 Probability Theory

Definition 2.2.1 An experiment is a process that may result in different individual outcomes, called *simple events*. The set of all possible occurrences is called sample space S. Sample space S is a collection of simple events $\{s_1, s_2, \ldots, s_n\}$.

Definition 2.2.2 An event E is a subset of sample space S. The probability of occurrence of E, $P(E) = \sum_{i=1}^{n} P(s_i)$, where $s_i \in E$.

Definition 2.2.3 If $E \in S$ is an event,

1. $0 \leq P(E) \leq 1$
2. $P(S) = 1$

3. $P(\phi) = 0$ (ϕ is empty set)

4. $P(\bar{E}) = 1 - P(E)$

Definition 2.2.4 Two events E_1, E_2 are *mutually exclusive* if $P(E_1 \bigcap E_2) = 0$.

Theorem 2.2.1 *For two mutually exclusive events,* $P(E_1 \bigcup E_2) = P(E_1) + P(E_2)$

Definition 2.2.5 Let E_1, E_2 be two events such that $P(E_2) > 0$. *The conditional probability* of E_1 occurring given E_2 has occurred (or simple E_1 given E_2) is given by,

$$P(E_1 \mid E_2) = \frac{P(E_1 \bigcap E_2)}{P(E_2)} \tag{2.1}$$

Definition 2.2.6 Two events E_1, E_2 are *independent* if $P(E_1 \bigcap E_2) = P(E_1)P(E_2)$.

Theorem 2.2.2 (Bayes' Theorem) For any two events E_1, E_2 with $P(E_2) > 0$),

$$P(E_1 \mid E_2) = \frac{P(E_1)P(E_2 \mid E_1)}{P(E_2)} \tag{2.2}$$

2.2.2 Binomial Distribution

Definition 2.2.7 Let n, k be non-negative integers. The *binomial distribution* $\begin{pmatrix} n \\ k \end{pmatrix}$ is the number of different ways of choosing k objects from n objects.

2.2.3 Entropy

Claude Shannon proposed the concept of *entropy* in his famous work, *A mathematical theory of communication* [83].

Definition 2.2.8 *The information entropy of a random variable* X*, that can take on possible values* $\{x_1, \ldots, x_n\}$ *is*

$$Ent(X) = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i) \tag{2.3}$$

The notion of entropy suggests uncertainty associated with a random variable. It is a quantification of information contained in a message. For example, tossing a coin has two outcomes with equal probability, so the entropy of a coin toss is 1 (substituting $p(x_1) = p(x_2) = 0.5$ in Equation 2.3). On the other hand for a

dice having six sides entropy is $-\sum_{i=1}^{6} \frac{1}{6} \log_2(\frac{1}{6}) = -\log_2(\frac{1}{6}) = 2.585$. Entropy is a tool for defining amount of information contained in one random variable about another random variable. The entropy of English language is between 1.0 and 1.5 bits per letter [81].

2.3 Cryptography

Cryptography deals with data security techniques such as encryption, signatures, hashing, oblivious transfer and secret sharing to name a few basic ones. Encryption refers to encoding messages with secret key(s) such that it is *hard* to decrypt the encoded message for anyone who does not have the key(s). Hardness here can be measured in terms of time/ effort/ memory one might need to decrypt messages by performing an exhaustive search of the key space. Generally, time and memory are the two major metrics used in determining the security of cryptographic schemes, also known as *cryptosystems*.

Cryptosystems can be classified into two categories based on the number of message bits processed at a given time.

1. *Stream Ciphers*: Encrypt individual characters/ digits/ bits of plaintext one at a time, using an encrypting operation which may vary with time.
2. *Block Ciphers*: Encrypt fixed sized groups of characters/ digits/ bits of plaintext at a time using an encryption algorithm.

Both stream and block ciphers can be used to build other cryptographic constructs such as MACs, hash functions and pseudorandom number generators. However, block ciphers are better understood and they offer greater security. The main appeal of stream ciphers is their efficiency. Cryptosystems are classified in the following two categories based on the keys they use at encryption and decryption ends.

1. Private key/ Symmetric cryptosystems
2. Public key/ Asymmetric cryptosystems

2.3.1 Private Key Cryptography

Private key cryptography or *symmetric cryptography* is the process of scrambling (*encryption*) plaintext (original message) into ciphertext (encrypted message) using

a secret key. The decryption from ciphertext to plaintext is performed using the same key. Advanced encryption standard (AES) [31, 32], a modified version of Rijndael, is the most widely accepted and used symmetric key cryptosystem. It was designed by Belgians cryptographers Joan Daemen and Vincent Rijmen as a part of the AES selection process and was announced on November 26, 2001. The key size in AES can be 128, 192, or 256 bits and it processes data blocks of 128 bits. Messages that are not a multiple of 128 bits are padded with one '1' followed by '0' bits to increase the message length to a multiple of 128 bits. The details of this cryptosystem, including the design and functioning can be found at NIST website [32].

Key-distribution is the major concern with symmetric cryptosystems. If two parties $\mathcal{A}$ and $\mathcal{B}$ who have never met before or transmitted a message to each other wish to communicate, then distribution of the cryptographic key is an issue.

2.3.2 Public Key Cryptography

Unlike private key cryptography, the public key cryptography applies two different keys for encryption and decryption. Each party that wishes to perform cryptographic operations needs to have a pair of keys, one is public and the other secret. The essential security requirement is that any body who knows the public key is not able to determine the secret key. Each party $\mathcal{P}$ has a pair of keys in this model, a private key and a public key. As the names suggest, only $\mathcal{P}$ knows his/her private key while $\mathcal{P}$'s public key is published in a public directory so that anyone can look it up. Messages encrypted with $\mathcal{A}$'s public key can be decrypted with $\mathcal{A}$'s private key and messages encrypted with $\mathcal{A}$'s private key can be decrypted with $\mathcal{A}$'s public key. If the sender uses the public key of the receiver, then the public-key cryptography is used for confidentiality. If the sender uses their secret key to create a cryptogram, then the public-key is used for authentication and the cryptogram is called a digital signature. Some of the most widely used public key cryptosystems are RSA, Rabin, ElGamal, McEliece and Merkle-Hellman. A brief discussion on RSA, which is arguably the most popular amongst them, is given below.

RSA Cryptosystem

Ron Rivest, Adi Shamir and Len Adleman invented this cryptosystem in 1977 [80]. The algorithm utilizes the hardness of factoring problem to encrypt messages. RSA

key generation algorithm is provided in Algorithm 3. RSA encryption and decryption algorithms are summarized in Algorithm 4 and Algorithm 5 respectively.

Input : Primes p, q
Output: Public key (n, e) and private key d
$n = p \times q$;
$\phi = (p-1) \times (q-1)$;
Select random e, $1 < e < \phi$, such that $gcd(e, \phi) = 1$;
Compute d, $1 < d < \phi$, such that $e \times d \equiv 1 \pmod{\phi}$;

Algorithm 3: RSA key generation

Input : Plaintext m $(0 \leq m < n)$, Recipient's public key (n, e)
Output: Ciphertext c
$c = m^e \pmod{n}$;

Algorithm 4: RSA encryption

Input : Ciphertext c $(0 \leq m < n)$, Recipient's private key d
Output: Plaintext m
$m = c^d \pmod{n}$;

Algorithm 5: RSA decryption

RSA algorithm is based on the computational hardness of factoring of the modulus. So the most obvious way of attacking the system is to try to factorize the modulus n. Most of the factorization algorithms (such as quadratic sieve) construct a quadratic congruence $X^2 = Y^2 \pmod{n}$ as the congruence is likely to produce the factors of n. However, certain design issues must be considered to avoid easy factoring by opponent. One of the major attacks on RSA is using non-trivial square roots of $1 \pmod{n}$. Assume that x is a non-trivial root of $1 \pmod{n}$, i.e., $1 < x < (n-1), x^2 \equiv 1 \pmod{n}$. Then $n \mid (x^2 - 1)$ or $n \mid ((x-1)(x+1))$, also $n \nmid (x-1)$, $n \nmid (x+1)$ since $1 < x < (n-1)$. Thus $\gcd(n, x-1) = p$ or q (using Euclid's theorem. As an illustration, take $n = 77$, we find two non-trivial square roots of 1 to be $34, 43$. Thus gcd(77,33) gives us $p = 11$ and thereby $q = 7$.

Digital Signatures

A digital signature is a number dependent on the message and a secret key known only to the sender of that message. In cases of dispute, a third party must be

able to resolve the issue without needing to know the sender's secret key. The following definition of digital signatures is taken from [110]. A digital signature scheme contains the following components:

- A security parameter k, which is chosen by the user when he creates his public and secret keys.
- A message space $\mathcal{M} \subseteq \{0,1\}^+$, which is the set of messages to which the signature algorithm may be applied.
- A signature bound B, which is an integer bounding the total number of signatures that can be produced with an instance of the signature scheme.
- A key generation algorithm G, which any user A can use on input 1^k (i.e. k in unary) to generate in polynomial time a pair (P_A^k, S_A^k) of matching public and secret keys.
- A signature algorithm σ, which produces a signature $\sigma(M, S_A)$ for a message M using the secret key S_A. Here, σ may receive other inputs as well.
- A verification scheme V, which tests whether S is a valid signature of message M using the public key P_A. (i.e., $V(S, M, P_A)$ will be true if and only if it is valid.)

Digital signatures serve the following purposes.

1. *Authentication*: It ensures that the sender of the message is in fact the real sender.
2. *Data integrity*: It gives the assurance that the message has not been tampered with since it is sent.
3. *Non-repudiation*: It provides a proof so the sender cannot deny sending the message.

In general, public key cryptography, and specifically RSA, provide an excellent way of implementing digital signature schemes.

In RSA public key digital signature scheme, the plaintext space and ciphertext space is both $\mathbb{Z}_n = \{0, 1, 2, \ldots, n-1\}$ where n is a product of two primes p, q.

Digital signatures can be created by reversing the roles of encryption and decryption because the encryption transformation is a bijection. The RSA digital signature generation algorithm and verification algorithm are provided in Algorithm 6 and Algorithm 7 [68], where R is a one to one mapping function from the message space to the signature space, called the *redundancy function*.

Input : Plaintext m, Sender's private key d, public key (n, e)
Output: Digital signature s for message M
$\tilde{m} = R(m)$;
$s = \tilde{m}^d \pmod{n}$;

Algorithm 6: RSA digital signature generation [68]

Input : Digital signature s, Sender's public key (n, e)
Output: Message authentication status
$\tilde{m} = s^e \pmod{n}$;
if $\tilde{m} \in \mathcal{M}_R$ **then**
| Recover message $m = R^{-1}(\tilde{m})$;
else
| Reject signature;
end

Algorithm 7: RSA digital signature verification [68]

2.4 Hash Functions

The following is a definition for a hash function (in the unrestricted sense) due to [68]:

Definition 2.4.1 *A* hash function $H : \{0,1\}^* \rightarrow \{0,1\}^n$ *is an operation that transforms an input x of arbitrary bit length to an output $y = H(x)$ of fixed bit length n (*compression*) and given x it is easy to compute $y = H(x)$ (*ease of computation.

IN addition to the above two properties of compression and ease of computation, the three desirable properties of *unkeyed hash functions* are provided below [68].

1. *Preimage resistance*: Given output y, it should be computationally infeasible to find x' such that $H(x') = y$.

2. 2^{nd} *preimage resistance*: Given input x, it should be computationally infeasible to find x' such that $H(x') = H(x)$. In this case, the user is provided with one input x and is free to choose the second input x'.

3. *Collision resistance*: It is computationally infeasible to find two inputs x, x' such that $H(x) = H(x')$. In this case, the user is free to choose both inputs.

Some examples of hash functions are MD4, MD5 and SHA-1. For more information, please refer to [68]. In particular, we will be concentrating on a special category of hash functions, known as *Message Authentication Code (MAC)*. The following definition of MAC is from [68].

Definition 2.4.2 *A* message authentication code (MAC) *algorithm is a family of function h_k, parameterized by a secret key k, with the following properties:*

1. *ease of computation* - for a known function h_k, given a value k and an input x, $h_k(x)$ is easy to computer. This result is called the MAC-value or MAC.

2. *compression* - h_k maps an input x of arbitrary finite bit length to an output $h_k(x)$ of fixed bit length n.

3. *computation-resistance* - given zero or more-text-MAC pairs $(x_i, h_k(x_i))$, it is computationally infeasible to compute any text-MAC pair $(x,h_k(x))$ for any new input $x \neq x_i$ (including possibly for $h_k(x) = h_k(x_i)$ for some i)

MACs are also known as *keyed hash functions*. In this work, hash function refers to keyed hash function or MAC, if the type is not specified. An example of MAC is MD5-MAC which is constructed using MD5 hash function. Details of MD5-MAC can be found in [68].

In this research, we use hash functions to determine whether a section of the digital object (for example, a tuple in a database relation or a sentence in a natural language document) will carry a watermark bit or not. The same hash function is re-used during detection phase to locate the sections of the digital object carrying the watermark bits.

2.5 Natural Language Documents

In this section, we give an introduction to natural language documents and, in particular, structure of the English language. Natural language documents are a

special category of text documents which have linguistic construction. They satisfy the rules of the grammar of a particular language. For example, a document written in the English language should satisfy the rules of English grammar. Natural language processing (NLP) deals with analyzing natural language documents for several purposes including text analysis, pattern matching, forecasting and more. Further information about NLP can be found in [66, 55]. There are several elements of a natural language, and some of them can be used to embed watermark bits by exploiting the redundancy in these languages. It is possible to convert a sentence in English to an equivalent sentence with little change to the meaning.

Since most of the documents on Internet are in English language, we will be discussing English language elements and structure in this work. The characteristics of other languages are similar with some variations. The components of English language are called parts of speech. Traditional grammar classifies words based on eight parts of speech, *verb*, *noun*, *pronoun*, *adjective*, *adverb*, *preposition*, *conjunction*, and *interjection* [3]. The parts of speech are explained below [4, 5, 6, 7].

1. A *verb* expresses existence, action, or occurrence. For example, in "Dad is *loading* the luggage", 'loading' is a verb.

   (a) The *main verb* is the most important verb in a sentence; without it, the sentence would not be complete. For example, in "He *took* my wallet", 'took' is the main verb.

   (b) *Auxiliary verbs* are words that can be used with the main verb to impart additional meaning, modify tense, or lay emphasis. Some of the auxiliary verbs are 'do', 'don't', 'does', 'doesn't', 'did', 'didn't', 'be' and 'have'. For example, in "He had taken my wallet", 'had' is the auxiliary verb.

2. A *noun* is used to name a person, place, thing, quality, or action and can function as the subject or object of a verb, the object of a preposition, or an appositive. Noun can be used as a subject or an object in a sentence [6]. For example, in "*Michelle* is brilliant", 'Michelle' is a noun.

   (a) The *subject* of a sentence is the noun, pronoun or noun phrase that precedes and governs the main verb.

   (b) The *object* of a verb is created, affected or altered by the action of a verb, or appreciated or sensed by the subject of the verb. For example,

in "*Colwin* played *soccer*", 'Colwin' is the subject while 'soccer' is the object.

3. A *pronoun* is a word that substitutes a noun or a noun-phrase (a noun-phrase is described shortly). For example, in "When John went to the pub, *he* was sober", 'he' is a pronoun referring to 'John'.

4. An *adjective* describes a noun. It describes the quality, state or action that a noun refers to. For example, in "This is a *ridiculous* requirement", 'ridiculous' is an adjective.

5. An *adverb* is a word that that describes the action of a verb, an adjective, another adverb, a noun or a pronoun. Basically, most adverbs tell you how, where, or when something is done. In other words, they describe the manner, place, or time of an action. For example, in "He *generally* arrives on time", 'generally' is an adverb.

6. A *preposition* is a word that links a noun, pronoun or gerund (*gerund* is a verb that acts as a noun, usually achieved by adding "*-ing*" to the verb) to other words. For example, in "He is going *to* London", 'to' is the preposition.

7. A *conjunction* is a word like and, but, when, or etc., which connects words, phrases or clauses. For example, in "He is eating pasta *and* she is having pizza", 'and' is the conjunction.

8. An interjection is a word or short phrase used in speech to gain attention, to exclaim, protest or command. Interjections can be used to show emotion such as surprise or shock. Interjections are often found at the beginning of a sentence, especially in speech, and are commonly followed by an exclamation mark or a comma. For example, in "*Ah*! That's a good book!", 'Ah' is the interjection.

Sometimes a group of words can provide the same functionality as a single word. Such groups are called *phrases*. Two important phrases are given below:

1. *Noun Phrases* are either a single noun or pronoun or groups of words containing a noun or pronoun that function as subjects or objects in sentences.

2. *Verb phrases* are groups of words that express action or state of being [7].

A *complement* is a word or words used after a verb to complete a predicate construction.

Patterns define the order in which the parts of speech can be arranged to construct a meaningful sentence. The five basic sentence patterns in English are as follows [8].

1. Subject + Verb
   Examples: *I play. Tanja eats. They walk.*
2. Subject + Verb + Object
   Examples: *I bought a car. Menno plays the guitar. They heard rumors.*
3. Subject + Verb + Complement
   Examples: *I am busy. Colwin became famous. They seem nice.*
4. Subject + Verb + Indirect Object + Direct Object
   Examples: *I gave her a present. Ms. Narayan teaches us English.*
5. Subject + Verb + Object + Complement
   Examples: *I left the stove on. We elected Colwin class monitor. They called her Didi.*

Tenses are used to convey the time frame of a particular event. The three tenses in English language are given below [8].

1. *Past*: expressing action that has occurred in the past, as in "*She rode the bike*"; "*He slept*".
2. *Present*: expressing action in the present time, as in "*She rides the bike*"; "*He is sleeping*".
3. *Future*: expressing action that has yet to take place, as in "*She will ride the bike*"; "*He will sleep*".

They can each be further broken into the following categories [8].

1. *Simple*
   (a) *Simple past tense* is used for past actions that happened either at a specific time, which can either be given by a time phrase (yesterday, last year, etc.) or understood from the context. For example, "*She taught computing*".

(b) *Simple present tense* is used to show permanent characteristics of people and events or what happens regularly, habitually or in a single completed action. For example, "*He drives a car*",

(c) *Simple future tense* is often called will, because we make the simple future tense with the modal auxiliary will. For example, "*He will drive a car*".

2. *Continuous*

(a) *Past continuous tense* is used for actions and states that were unfinished at a certain time in the past or to stress the duration of something. For example, "*She was teaching computing*".

(b) *Present continuous tense* is used for actions that have begun but not finished. It can also be used to talk about future arrangements. For example, "*He is driving a car*",

(c) *Future continuous tense* is used for actions that will be unfinished at a certain time in the future, or for things that will happen in the normal course of events, rather than being part of your plans and intentions. For example, "*He will be driving a car*".

3. *Perfect*

(a) *Past perfect tense* is used for actions that happened before related past events or times. For example, "*When they chose her, she had taught computing*".

(b) *Present perfect tense* is used for unfinished past actions. For example, "*He has driven a car for two years*".

(c) *Future perfect tense* is used for actions to be completed before a specific future time, but the exact time is unimportant. For example, "*He will have driven a car in five years*".

4. *Perfect Continuous*

(a) *Past perfect continuous tense* is used for actions that were unfinished when another action, etc., took place. For example, "*He had been driving a car*".

(b) *Present perfect continuous tense* is used to emphasize the duration of a recent past activity. It can also be used for actions that began in the past and are still going on now. For example, "*He has been driving a car*".

(c) *Future perfect continuous tense* is used used for actions that will be unfinished, but have reached a certain stage. For example, "*He will have been driving a car for three years by then*".

Syntax and *semantics* should be addressed while structuring a document in the English language document.

1. *Syntax*: This is the order in which the components within a sentence should be arranged to make sense. For example, there are many possible syntaxes in English language. One of the simplest of such valid syntaxes is given below [92].

$$(NP_1) \rightarrow (\text{linking verb}) \rightarrow (NP_2)$$

where NP_1 and NP_2 are noun phrases. A sentence that satisfies the above syntax is *"Dogs chew bones"*.

2. *Semantics*: Semantics refer to the meaning the document conveys to the reader. Depending on the language, the same text message can have multiple meanings. Such languages are called *ambiguous languages*. For example, *The pencil is rolling near the eraser, can you pass it to me?* can either mean that you are being asked to pass the pencil (with higher probability) or the eraser (with lower probability). Ambiguous languages are beneficial for our purpose since we can convert a text message in one syntax to another text message with different syntax but with similar semantics.

2.6 Software

Software is a set of programs working together to achieve a pre-defined set of tasks. The programs that form a software are sets of instructions written in the same or different languages. For further information about design of programming languages, please refer to [37, 82, 38, 41]. The categories of programming languages area are as follows.

1. *Low-level languages*: Languages that interact directly with the computer hardware. Following are the sub-categories of low-level languages.

   (a) *Machine-level languages*: Programs are strings of 0s and 1s where each bitstring pattern has a particular meaning.

   (b) *Assembly-language languages*: Program instructions are from a set of predefined operations such as *move*, *store*, etc.

2. *High-level languages*: Programs are written in human-understandable terms and then translated to low-level language. High-level languages are further categorized as the following.

   (a) *Compiled languages*: Program is translated to machine-code, which is saved separately . This process is called *compilation*, and is followed by running the compiled program, called execution. For subsequent executions of the program, compilation need not be done, thus saving time.

   (b) *Interpreted languages*: Program is interpreted instruction by instruction and compilation is not performed. Programs written in these languages take longer to execute since they have to be interpreted each time they are executed.

The most commonly used programming languages are C, C++, and Java and we design our watermarking schemes with these languages in mind. While C and C++ facilitate direct memory manipulation using pointers, Java prohibits memory manipulations to provide greater security. The common characteristics of these three languages are as follows.

- *Basic data types*: Data stored in these languages can be of integer type (boolean, integer), floating-point type (float, double), symbolic type (char, string).

- *Sequential execution*: Under normal circumstances, the instructions are executed sequentially and this order can be manipulated by using *goto* (not recommended), *break*, and *continue* operations.

- *Control Statements*: There are two kinds of control statements in these languages,

1. *Repeat Statements*: To perform the same operation(s) multiple times, a programmer can use one of the three repeat statements *for*, *while* and *do-while*. These statements are also called loops because of their inherent nature.
2. *Conditional Statements*: This refers to selecting a course of action depending on some condition. We can deploy conditional statements such as *if*, *if-else* and *switch*.

- *Functions/ Methods*: For tasks that need to be repeatedly performed with same or different parameters, programmers can write functions or methods. These functions are called from a particular statement and after completing execution, they return back to the calling statement with some return value or message.

- *User-defined data types*: C enables grouping of multiple elements of different types together as *structures*. C++ and Java have *classes* that provide the same facility as structures. But a programmer can do much more in classes, such as including functions or methods, defining friend functions and inheritance (discussed shortly below).

- *Arrays*: A collection of elements of similar type can be stored in a set called arrays.

Object Oriented Paradigm (OOP) provides a framework for software development. The fundamental principles of OOP [59] are as follows.

1. *Encapsulation*: Encapsulation is hiding data implementation by restricting access to *accessors* and *mutators*. An accessor is a method that returns values of one or more data members the object itself. However, accessor methods are not restricted to attributes but can be any public method that gives information about the state of the object. Mutators are methods, generally public, that are used to modify the state of an object by modifying attribute values, while hiding the implementation of exactly how the data gets modified. Mutators are another portion of the encapsulation property, except this time it is the set method that lets the caller modify the member data behind the scenes.

2. *Abstraction*: Abstraction represents a model or some other focused representation for an actual entity. Data abstraction is the development of classes,

objects, and types in terms of their interfaces and functionality, instead of their implementation details. It is the development of a software object to represent an object we can find in the real world. Encapsulation hides the details of that implementation.

Abstraction is used to manage complexity as software developers use abstraction to simplify complex systems into smaller and manageable components. Programmers are constantly aware of the functionalities to be provided by the subsystems that are in the development stage. Hence, programmers are not burdened by considering the ways in which the implementation of later subsystems will affect the design of earlier development.

One of the key members of the development team for Object Oriented Technology, Grady Booch, defines abstraction as "An abstraction denotes the essential characteristics of an object that distinguish it from all other kinds of object and thus provides crisply defined conceptual boundaries, relative to the perspective of the viewer." [17]

3. *Inheritance*: Two objects can have either of the following relationships.

   - "*has a*",
   - "*uses a*" or,
   - "*is a*"

   "*Is a*" is the *inheritance* way of object relationship. Take the example of a library system. A library lends books, magazines, audio cassettes, and more. At a fundamental level, all four types represent *assets* of the library that can be loaned out to people. However, even though the four types can be viewed as belonging to the same category, they are not *identical*. They have some differences as well; a book has an ISBN and a magazine does not, audio cassette has a play length and video microfilm cannot be checked out overnight.

   Because of such underlying differences, each of these assets of the library should be represented by its own class definition. But the common characteristics can be put in a *superclass*, or, *base class*. Without inheritance though, each class must independently implement the characteristics that are common to all loanable assets. All assets are either checked out or available for checkout. All assets have a title, a date of acquisition and a replacement cost. Rather

than duplicate functionality, inheritance allows you to inherit functionality from another *base class*.

4. *Polymorphism*: Polymorphism means one interface, different behavior. Polymorphism refers to having multiple methods all with the same name, but slightly different functionality. Some of the common methods that need polymorphic behavior are $add, areEqual, print, sort$ and $isEmpty$. There are two basic types of polymorphism: *overriding* (also called run-time polymorphism), and *overloading* (also called compile-time polymorphism). For method overloading, the compiler determines which method will be executed when the code is compiled. Which method will be used for method overriding is determined at runtime based on the dynamic type of an object.

Object Oriented Principles can be of assistance for hiding the watermark inside the class definition by utilizing the access modifiers and hiding the watermark in inaccessible areas of the software.

2.7 Databases

Databases refer to structured data being stored in computer systems. A further refined term, relational database refers to a special class of databases, that are arranged in relations or tables. Data can be accessed and the output can be rearranged, filtered, and manipulated without having to modify the tables. Relational databases are collections of relations or tables where each table contains one or more columns (attributes) and rows (tuples). Information about every entity is stored in a different tuple and attributes specify the type of information.

Each table should have a special attribute called *primary key*. If the table does not have an inherent primary key, an extra attribute is appended to the table containing unique values for each tuple (similar to a serial number). The primary key must be unique for every tuple and is used to identify a particular entity. For example, passport numbers serve as primary key for the relation "Passengers".

Two tables t_1 and t_2 are said to be *related* if they have at least one attribute in common. For most purposes, the common attribute A is the primary key in one of the tables, say t_1. The attribute A is called a *foreign key* in the other table (t_2).

If there is no attribute guaranteed to be unique for each tuple, then we cannot define a primary key directly. In this case, we create a primary key using one of the

following methods.

1. A combination of several attributes serves as a primary key. Such a primary key is called *composite key*. For example, {*Room,Day,Time*} is a primary key for a *Lecture* relation.

2. The database management software adds an extra attribute to the table called, such as *ID*.

Query is the most important process in a database system. Although there is no authoritative definition of a query, it can be termed as *performing a operation on a relational database to read data from or write data to or update data in one or more database tables*. The data to be read, written or updated can be selected using conditions in a query. Standard query language (SQL) provides a framework for querying a relational database. Following is the basic structure of an SQL query:
SELECT <*attribute(s)*>
FROM <*table(s)*>
WHERE <*criteria*>

This section has described the basic building blocks of a database system. Please refer to [33, 35] for more details.

Chapter 3

Overview of watermarking

In this chapter, we introduce the background of paper-based watermarking that was used as early as 1282 A.D., and describe the current digital watermarking scenario. We also discuss approaches to, and techniques for, digital watermarking. Several important watermarking schemes are described and analyzed as well, in order to gain an understanding of the current watermarking standards and opportunities.

Historically, the paper on which documents were written or paintings were drawn contained inherent marks that helped identify the paper itself, its region, owner's identity and also time period. Paper-based watermarking refers to adding an impression incorporated in the paper making process. The impression shows the name of the paper and/or the company logo. Watermarking is also applied to detect any manipulations on the paper document - if someone tries to modify the text, the watermark would be affected as well. Digital watermarking can be perceived as a technology-specific instance of watermarking that achieves watermark embedding and detection in digital multimedia objects.

In 1804, G. Fischer, a German anatomist, entomologist and paleontologist, published the list of watermarks, *Beschreibung einiger typographischen Seltenheiten* [91], in which he stated that the oldest watermark in existence was on paper produced in 1301. However, C. Briquet discovered a watermark printed on paper produced in 1282 [20]. Watermarks from the 14th century have also been mentioned in historical articles [34]. The oldest known watermark kept in Australia is Bishop's Crosier from the 16th century [93], shown in Figure 3.1. We provide some illustrations of watermarks from 16th century to the 20th century. Figure 3.2 and Figure 3.3 show currency watermarks [10]. Figure 3.4 shows a Spanish document containing

watermark from 17th century [9]. A magnified view of the watermark is given in Figure 3.5.

Figure 3.1: Bishop's crosier (Australia), 16th century

Figure 3.2: Watermarks in Australian currency bill

Figure 3.3: Watermarks in German currency bill

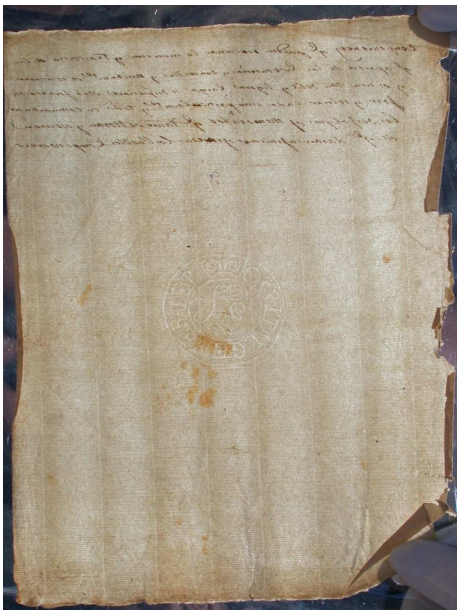

Figure 3.4: Watermark in Spanish document from 17th century

Figure 3.5: Magnified view of watermark from Figure 3.4

The basic purpose of digital watermarking still remains the same as that of traditional paper-based watermarking. However, computer science and telecommunication technology allows watermarking to be applied to a whole range of multimedia objects such as image, audio, video, databases, software and text documents. Watermarking techniques have also improved considerably to survive various possible attacks such as resizing (in images) or modifying numerical values (in databases).

3.1 Approaches to Watermarking

There are many watermarking schemes within the industry; however, the watermarking method that is used depends upon the purpose of watermarking the object in the first place. Following are the categories of watermarking, containing both the method of watermarking along with the intended purpose.

- *Perceptibility*: The degree of watermark perceptibility and visibility.

    1. *Perceptible Watermarking*: When the purpose of watermarking is to discourage someone from copying a multimedia object and reduce its quality deliberately, the watermark is made quite visible and this technique is called *perceptible watermarking*. This type of watermarking is effective only if the user is law-abiding and we can assume that (s)he would not copy the object if aware of the copyright status of the object.

2. *Imperceptible Watermarking*: In most cases, not only do we want to make the user aware of the ownership, we also desire to embed an *invisible* mark in the object. Following are the advantages of imperceptible watermarking.
   - Watermark does not reduce the quality of the multimedia object.
   - It is difficult for the attacker to locate the watermark, thereby making the watermarking more robust.

   The user can be an active attacker where (s)he either copies, modifies, or redistributes the multimedia object *knowing* that it is copyrighted. This category of watermarking is called *imperceptible watermarking*. An imperceptible watermark should be invisible to the user and should not significantly deteriorate the quality of the multimedia object.

- *Purpose*: What is the objective of watermarking?

  1. *Tamper-proofing*: The purpose of embedding this watermark type is to detect tampering with the multimedia object.
  2. *Owner-identification*: The objective of watermarking is to establish ownership over a multimedia object.

- *Robustness*: Refers to degree of robustness and resilience against attacks. This criterion also measures the effectiveness of watermarking as tamper-proofing or owner-identifying.

  1. *Fragile Watermarking*: Fragile watermarks are easily destroyed by minor modifications on multimedia objects. Therefore, they find applications in tamper-detection where the owner might want to detect manipulation on the object.
  2. *Robust Watermarking*: Robust watermarks are embedded to establish ownership and therefore are effectively robust and resilient against different attacks. Ideally, even major or extensive attacks should not destroy the watermark.

- *Detectable/ Extractable*: Where the actual watermark is extracted or its presence simply detected.

1. *Extractable Watermarking*: Meaningful watermarks (for example *"©: IBM Corporation"*) are embedded in multimedia objects. The watermark can contain information like owner identity and time stamps. The same watermark when extracted from the multimedia object establishes ownership.
2. *Detectable Watermarking*: The embedded watermark is only detected as *present/absent*. Thus, information like owner identity or time stamps, are not required for detectable watermarks.

- *Requirements for watermark detection*: Does the watermark detection algorithm need inputs other than the watermarked content and some secret key?

  1. *Non-Blind Watermarking*: The original multimedia object (or a part of it) is required in addition to the watermarked document during detection.
  2. *Semi-blind watermarking*: The bits modified in the original object are supplied to the detection algorithm along with the watermarked object.
  3. *Blind Watermarking*: Blind watermarking does not require the original object during detection. The only objects needed for watermark detection are the watermark object and a secret key known to the owner.

- *Reversibility*: Ability to reverse a watermarking procedure and thereby recover the original content from the watermarked content..

  1. *Irreversible*: Once the watermark is embedded, the original object cannot be generated from the watermarked object.
  2. *Reversible*: The original object can be regenerated from the watermark object using the secret key and the watermarked object. This category of watermarking is rarely discussed in the literature, but we have identified reversibility as a solution to secondary watermarking attacks. We present and analyze this solution in Chapters 6, 7.

3.2 Text and Natural Language Watermarking

Text and natural language watermarking have become commercially important with the ever-growing rise of business interest in digital text content such as e-books, blog entries, document repositories and online distribution of literature. In recent

years, we have seen several lawsuits involving copyright infringement of textual objects. Therefore, it becomes increasingly crucial to place imperceptible and detectible/extractable copyright marks in natural language documents. Firstly, we differentiate between two terms commonly (yet incorrectly) perceived to be interchangeable; *natural language watermarking* and *text watermarking*.

Text Watermarking	**Natural Language Watermarking**
Text need not have meaningful structure.	Text should have a linguistic structure.
Text need not follow grammatical rules.	Text needs to follow grammatical rules.
Susceptible to reproduction attacks.	Reproduction attacks are ineffective.
Watermark embedded in formatting.	Watermark embedded in structure.

In one of the early works in format-based text watermarking, [50] inserts watermark in a text document by changing inter-word spacing. The research provides a good indicator of the capabilities and limitations of general format-based text watermarking schemes. It calculates average inter-word spacing for different lines, $S(i)$, then for each line, a watermark is determined by a sine wave $W_n = C_1.a_1.\sin(\phi_1(n-p)+\varphi_1)$, where,

- n is the line number
- p is the index of the fist line in the workplace after which a watermarking sine wave will reside
- W_n represents the desired watermark of a text line with an index of n
- ϕ_1 and φ_1 are the radian frequency and initial phase angle of the sine wave respectively
- C_1 is a constant determining the amplitude of the sine wave; and
- a_1 is the mean of $S(i)$s for different lines.

The new average inter-word spacing, S'_{an}, is calculated as $S'_{an} = S_{an} + W_n$, where S_{an} is the original inter-word spacing. Finally the words in each text line are modified by applying the parameters obtained above with document characteristics. This method can be implemented for both private and public watermarking. The detailed insertion process is shown in Algorithm 8. The detection is non-blind and

re-inserts the watermark and compares the result with the watermarked copy, and if they match, it detects the watermark successfully. The following are the two major drawbacks of the watermarking scheme.

1. The scheme is vulnerable to reproduction (photocopying), reformatting and resizing attacks.

2. The scheme requires the original document to detect the watermark and hence is a non-blind watermarking scheme.

Input : Document D, Lines $\{S_1, S_2, \ldots, S_n\}$, secret key K
Output: Watermarked document
forall *lines* S_i *in document* **do**
  d=number of words in line S_i;
  if $d > K$ **then**
    S_t=total inter-word space in line S_i;
    Pxl_i=width of line in pixels;
    Average inter-word space $S_{a_i} = \frac{S_t}{d-1}$;
    $W_i = C_1.a_1.\sin(\phi_1(i-p)+\varphi_1)$;
    $S'_{a_i} = S_{a_i} + W_n$;
    $S_{tc} = \frac{S'_{a_i} - S_{a_i}}{d-1}$;
    if $S_{tc} \geq 0$ **then**
      Shrink in word width $ES_i = \lfloor S_{tc}.\frac{Pxl_i}{\sum_{i=1}^{d} Pxl_i} \rfloor$;
    else
      Expansion in word width $ES_i = \lceil S_{tc}.\frac{Pxl_i}{\sum_{i=1}^{d} Pxl_i} \rceil$;
    end
    Interval $Iv_i = \lfloor \frac{Pxl_i}{ES_i} \rfloor$;
    Duplicate/Remove vertical lines at interval Iv_i;
  end
end

Algorithm 8: Watermark insertion changing inter-word spacing

Amongst synonymy-based watermarking schemes, a fundamental model is proposed in [56]. This model describes the general scenario in synonymy-based text watermarking, where words and phrases are replaced by synonyms, thereby introducing *tolerable* distortion in the text document.

In the field of synonymy-based text watermarking, the notion of *absolute* synonyms and *equivalences* is put forth in [15]. According to the study, the terms are defined as follows.

- *Absolute synonyms* are a set of words that can replace each other in **context** without **any** change in meaning. For example {*sofa*, *settee*}.
- *Equivalences* are a set of words that can replace each other with some degree of change. Equivalences cannot replace each other unconditionally. Abbreviations and full forms are examples of equivalences as in {*UK*, *United Kingdom*}, since you cannot replace one by another in the sentence "UK stands for United Kingdom".

Researchers exploit the absolute synonymy to embed a message in [15]. Based on a shared dictionary, first synonym pairs are generated for the document. Next, based on the message, corresponding synonym group is chosen.

Let the message be M_d, and p word pairs $w_0, \ldots w_{p-1}$ are identified with w_i having k_i corresponding synonym pairs. In this first step, the message is encrypted using a secret key K to M. The first word pair is changed to one of the corresponding synonym pair depending on the message digit M_0. If M_0 is greater than the number of pairs available, then it is changed to the synonym pair $M_0 \pmod{k_i}$ and the rest of the message (M_0/k_i) is forwarded to the next pair. For example, let 5 word pairs exist with number of synonym pairs 6,3,8,4, and 5. If the message is 231, it cannot be contained in the first pair. The first pair is changed to synonym number $231 \pmod 6 = 3$ and the remaining message 231/6=38 is forwarded. The second pair cannot contain this message so it is changed to synonym $38 \pmod 3 = 2$ and remaining message 38/3=12 is forwarded. This process continues till the entire message is embedded. The message embedding is formally represented below with Algorithm 9 describing the embedding process. The detection process works in the exact way to extract the encrypted mark M and decrypts it with K to reveal the message M_d.

The most serious weakness of the scheme is that the selection of words that carry the mark is not based on a secret key. The secret key is actually used to encrypt the message as the first step of insertion process, and to decrypt the message as the last step of the detection process. Hence, anybody can extract the encrypted message (since the synonymy dictionary is publicly available) and destroy it.

Following are the disadvantages of this category of watermarking scheme.

Input : Message M_d, Document D, Key K
Output: document D_m containing message M_d
Encrypt M_d with K to get M;
Get word pairs $w_0, \ldots w_{p-1}$ from D;
$i = 1$;
while *watermark to be embedded* **do**
  Synonyms for $w_i = \{sp_1, \ldots, sp_{k_i}\}$;
  if $i < p$ **then**
    $n_i = M_{i-1} \pmod{k_i}$;
    $M_i = \frac{M_{i-1}}{k_i}$;
    $w_i = sp_{n_i}$;
  else
    $n_p = M_{p-1}$;
    $w_p = sp_{n_p}$;
  end
  $i = i + 1$;
end

Algorithm 9: Watermarking using collocationally-based synonymy

- Watermarking scheme is non-blind since you need a shared dictionary at embedding and receiving end.
- Watermark detection would fail upon reshuffling the document.
- Watermarking scheme is not reversible so that the original document cannot be extracted from the watermarked copy.
- Synonym pairs are not weighed which results in significant loss of meaning.

Atallah et. al present the first significant research works in natural language watermarking in [14]. The main significance of their work is that it illustrates the fundamental differences between format/synonymy-based watermarking and semantic-based watermarking and the advantages of the latter approach against format/synonymy-based text watermarking. It provides a scheme for inserting a watermark in a natural language document. If a selected sentence does not yield the bit(s) needed it to yield, an attempt is made to generate the correct bit sequence by transforming the sentence without any significant meaning change. Standard transformations described in post-Chomskian generative syntax are tried for this purpose. There are few but very productive syntactic transformations that change

the structure of a sentence while preserving its overall meaning. These are given below.

- Adjunct movement
- Clefting
- Passivization

Let the original sentence be "*the dog chased the cat*". The following set of notation is used during the parsing of sentences, which is done using Link Parser [1].

- *S* connects subject-nouns to finite verbs
- *NP* represents *Noun Phrase*
- *VP* represents *Verb Phrase*
- *ADVP* represents *Adverbial Phrase*
- *SBAR* represents *Complement Sentence*
- *WHNP* represents *Relative Pronoun Phrase* (pronoun that points to another noun in the sentence)

Original sentence:

```
(S (NP the dog)
    (VP chased
      (NP the cat)))
```

Adjunct Movement: An adjunct, like a prepositional phrase or adverbial phrase, can occupy several well-defined positions in a sentence. For example, the adverbial phrase often can be inserted in any of the positions marked by ADVP, and when originally found in one of these, can be moved to any of the others:

```
(S (ADVP often)
    (S (NP the dog)
```

```
(VP (ADVP often)
   chased
      (NP the cat)
         (ADVP often))))
```

Clefting: Clefting adds emphasis to the sentence and can most easily be applied to the mandatory subject of a sentence. In this case, pointing to the dog.

```
(S (NP it)
    (VP was
      NP (NP the dog)
         (SBAR (WHNP that)
            (S (VP chased
               (NP the cat)))))))
```

Passivization: Any sentence with a transitive verb can be passivized. Identifying the syntactic structure of such a sentence is simple, even in the output of a very basic syntactic parser. A transitive verb has a subject *NP1* and an object *NP2* which is the complement that occupies the sister-node of the verb. Ignoring factors like tense, aspect, number, and modal auxiliaries (which are easily implemented), the following is the passive sentence generated out of this input, where *PP* stands for *prepositional phrase*.

```
(S (NP the cat)
    (VP was
      (VP chased
         (PP by
            (NP the dog)))))
```

A change to the syntax of a sentence can also be achieved through sentence-initial insertion of semantically empty *transitional* phrases like generally speaking, basically, or it seems that.

```
(S (NP it)
    (VP seems
      (SBAR that
        (S (NP the dog)
          (VP chased
            (NP the cat))))))
```

The sentences are ranked in the document according to their size and the first chunk of the watermark is inserted in the sentence following the least-ranked sentence, s_{i-1}. A *marker* is a sentence whose successor in the sorted sentence set is chosen to contain the watermark. The watermark bits are stored in the binary string B_i corresponding to s_i, until relevant bits of B_i match the desired value. Finally the position of s_i is modified in the sorted sentence sequence. The pseudo-code is provided in Algorithm 10.

A summarization of the current text and natural language watermarking scenario is provided in Table 3.1 containing security offered by text watermarking schemes from the three classes (format-base, synonymy-based, and semantics-based).

3.3 Software Watermarking

Software watermarking is the process of watermarking the source code so that limited manipulation of the source would not alter the watermark beyond recognition, and thereby preserving owner identity. It is fairly complex to re-engineer the source code from executables but it cannot be entirely rejected. Owner identification in the software industry becomes even more crucial to ensure that open source software remain freely available since an attacker might try to modify an open source code and create an application with differing user interface but essentially the same processing algorithms. This application may be made available for a cost in the

Input : Document D, Watermark W, Key p (prime number)
Output: Watermarked Document D_w
Watermark $W = \{w_1, \ldots, w_m\}$; `// ` w_i `is a bit`
Sentences in document $\{s_1, \ldots, s_n\}$;
For all i, size of s_i ¡ size of s_{i+1};
$j = 1$;
while *watermark to be embedded* **do**
  while s_{j+1} *has been chosen as a marker* **do**
    Remove s_j from S;
    $j = j + 1 \pmod n$;
  end
  s_j chosen to contain watermark;
  Marker sentence is s_{j-1};
  T_j is the syntactic tree representation of s_j;
  Number nodes of T_j according to pre-order traversal;
  forall *nodes* $i \in T_j$ **do**
    if $i + H(p)$ *is a quadratic residue modulo* p **then**
      $i = 1$;
    else
      $i = 0$;
    end
  end
  B_j is bitstring generated from post-order traversal of T_j;
  B_j is called node encoding of s_j;
  Insert β bits in the LSB of B_j to get B'_j;
  Apply transformations to s_j to get s'_j having node encoding B'_j;
  Move s_j to maintain sorted order of S;
end

Algorithm 10: Natural language watermarking [14]

Attack	Format-based [50]	Synonymy-based [56]	Synonymy-based [15]	Semantics-based [14]
Reformatting	Insecure	Secure	Secure	Secure
Data deletion	Insecure	<Secure>	<Secure>	<Secure>
Data addition	Insecure	<Secure>	<Secure>	<Secure>
Data Reordering	Insecure	<Secure>	<Secure>	<Secure>
Synonym-substitution	Insecure	Insecure	<Secure>	Secure
Rephrasing	Insecure	Insecure	Insecure	<Secure>

(<Secure> implies partial security)

Table 3.1: Comparative study of text watermarking schemes

software market.Thus the attacker is effectively profiting from someone else's work which was, in the first place, aimed at providing convenience to the general user without any monetary benefits in return.

3.3.1 Taxonomy of Software Watermarking

Software watermarking is categorized as follows.

1. *Static vs. dynamic watermarks*: In static watermarking, watermark is inserted in the data section or the physical code of the program, while in dynamic watermarking; it is embedded in the execution of the program.

2. *Visible vs. invisible watermarks*: Watermarks are explicitly placed inside the software in visible watermarking, but in the case of invisible watermarking, they are generally generated during the execution of the program in the case of invisible watermarking.

Software watermarking schemes can also be classified based on the elements exploited in order to embed the watermark into the following sub-categories.

1. *Graph based software watermarking*: The watermark is encoded in the control flow graph (CFG) of the software. This class of watermarking exploits redundancies in software execution such as branch instructions and function or methods calls. If a program C with CFG is modified such that the program changes to C' but there is no effect on the output of the software, we can say that the two programs are equivalent from the users' perspective. This happens when intermediate nodes are added to the execution path but the end-result is preserved. Let b_1 be a sequential code block that branches to another block b_2. If we add instruction(s) i_{med}, such that b_1 branches to i_{med}, which in turn branches to b_2, there is no net effect on the programs execution. However the CFG changes and the watermark is embedded in its encoding. Figure 3.6 illustrates this pseudo-code insertion. Watermarking algorithms from this family are presented in [90, 29, 28, 27, 26].

2. *Register based software watermarking*: Register allocation can be enforced and manipulated to embed the watermark. If two variables are not used simultaneously at any given time during program execution, then the same

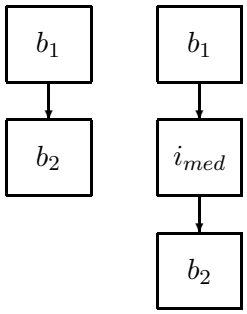

Figure 3.6: Inserting intermediate code without effecting output

register can be allocated to store these variables. This situation is exploited by several watermarking models [75, 76, 79, 78, 77, 95, 57, 70, 106].

3. *Thread based software watermarking*: These schemes alter the threads that execute particular sections of the code. The obvious drawback of such schemes is to have a multi-threading environment in which the program is written. These schemes are not extremely popular due to their language-dependency. One such scheme is proposed in [72].

4. *Obfuscation based software watermarking*: In this category, the watermark is either statically embedded in dummy methods/ functions/ instructions or the classes are manipulated by splitting or merging multiple classes to encode the watermark in the class encoding. Models from this category [39, 69, 74, 51] are described further in this chapter.

3.3.2 Attacks on Software Watermarks

There are two categories of attacks on any watermarked multimedia object; generic and multimedia-specific. Generic attacks reorder, add, delete, and modify data while multimedia-specific attacks exploit the object characteristics to identify the watermark location and delete the watermark. The attacks on watermarked software that are software specific are given below.

1. *Decompilation/ recompilation*: In this attack, the attacker decompiles the machine code into a high level code and recompiles the program.

2. *Variable restructuring*: The attacker might change variable names, variable scopes and variable ordering to erase the watermark if embedded in variable structure.

3. *Synonymous instructions substitution*: Instructions in programming languages can usually be replaced by other instructions which perform the exact task. For example, in C/C++/Java, a programmer can replace *for* loop with *while* or *do-while* loop. The attacker can change such instructions in hope to erase the watermark.

4. *Addition/ deletion/ displacement of dead code*: If the watermark is embedded in dead code (code that never gets executed), an attacker can modify such dead code to erase the watermark.

5. *Re-ordering of independent code blocks*: If there are multiple code blocks that are mutually independent, they can be re-ordered without having an adverse effect on the output of the program. But a watermark that is embedded in such an ordering would be destroyed by such operation.

6. *Checking unreachable code*: If there is unreachable code in the program, the attacker can delete it. If the watermark detection algorithm uses input parameters such as program size, the watermark will not be detected.

7. *Register re-allocation*: Assume that watermark is hidden in the ordering of registers holding the variables. An attacker can re-order this register allocation and thereby modify the watermark encoding.

8. *Re-assigning thread sequence*: Certain software watermarking schemes embed watermark in the sequence in which threads are executed. This can be attacked by reassigning thread priorities and sequence of execution.

9. *Class manipulation*: If the watermark is embedded in the obfuscation of classes, then the classes can be merged or split to eliminate the watermark.

These attacks need to be dealt with, while maintaining semantic correctness of software and preserving basic principles of programming (such as encapsulation, data abstraction for object oriented programming).

3.3.3 Review of Software Watermarking Schemes

Amongst the different categories of software watermarking, the most general category is the branch-based software watermarking since it targets the very basic characteristic of a software - branching. Each software can be interpreted as a graph and thus it is a generic approach of watermarking that can be extended to almost any source code written in any language. Hence, we first describe research that deal with graph-based approach to watermarking.

The ground work in graph-based software watermarking has been carried out by Venkatesan et al in [90]. The basic principle is to convert the software and the watermark code into digraphs. The software to graph encoding works as follows.

1. The nodes of the graphs are basic blocks of codes (sequential code).
2. The edges of the graphs are function calls from one block to another.

New edges are introduced between the two graphs implemented by adding function calls between the software and watermark code. This scheme lacks any form of error-correcting codes and is susceptible to reordering of instructions, adding of new function calls. Another problem in the scheme is that the random walk mentioned in the work is not truly random. The node to be visited next is determined based on probability. This also gives away information to the attacker. Alternatively, a pseudo-random permutation of the nodes which will be visited can be generated. Collberg and Thomborson later presented other models based on improvement over this scheme in [28, 27].

Several graph encodings can be used to encode the software and the watermark code. Some of these encoding have been provided in [29], and are briefly described below:

1. *Radix-k encoding*: This is a circular linked list representation, where a null-pointer encodes 0, self-pointer means 1, pointer to next node means 2 and so on. A list of length m can encode an integer in the range $[0, (m+1)^m - 1]$. For $m = 255$, 2040 bits can be encoded. There are two pointers in every node; the right pointer holds the location of the next field as in a normal linked list, while the left pointer encodes a base-k digit. In a Radix-5 encoding, the number 266 would be encoded $2 \times 5^3 + 3 \times 5^1 + 1 \times 5^0$ and therefore, we would need six nodes; one header node (n_0), node n_1 pointing to the next node n_2

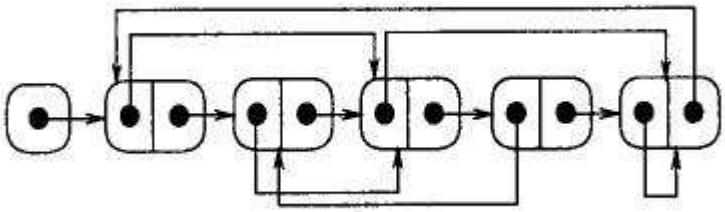

Figure 3.7: $61 \times 73 = 3.6^4 + 2.6^3 + 3.6^2 + 4.6^1 + 1.6^0$ in Radix-6 encoding [29]

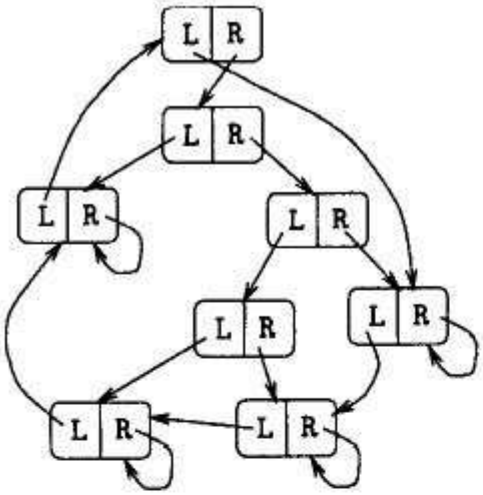

Figure 3.8: Planted Planar Cubic Tree [29]

and thereby encoding the digit 2, n_2 pointing to node n_5 (leaving n_3, n_4), thereby encoding digit 3 and n_5 self-pointing and encoding digit 1. Figure 3.7, taken from [29], illustrate a radix-6 encoding that corresponds to value 4453 (61×73).

2. *Planted planar cubic trees (PPCT)*: PPCT is a binary tree with an additional node, called the header, pointing to the root. All the leaf nodes form a circular linked list from the *rightmost leaf* $\rightarrow$ *leftmost leaf* $\rightarrow$ *header* $\rightarrow$ *rightmost leaf*. Each node has two pointers, l and r. Right pointer of a leaf node points to the node itself ($right(v_{leaf}) = v_{leaf}$). Figure 3.8, taken from [29], provides an illustration of a PPCT.

3. *Repetitive PPCT*: Each node of a PPCT is made redundant to get n nodes

from each node. Successful recovery of majority of the n nodes results in correct recovery of original node.

A code that builds the watermark graph is embedded in the software. This code is executed upon a special input I (acting as the secret key).

PPCTs are re-used in [28] and it also introduces reducible permutation graphs (RPG), having better error-correcting properties for graph embedding. An RPG has the following properties.

1. *header node* has in-degree zero, out-degree one. Any node in the graph can be reached from here.
2. *preamble* nodes follow the header node and have no forward edges, except those used in graphs Hamiltonian path.
3. *body* are edges between these nodes encode a self-inverting permutation.
4. *footer node* has out-degree zero. It is reachable from any node in the graph.

Modifications to the above mentioned work on RPG, capacity improvement and security analysis is performed in [27]. In [26], Collberg et al. propose two watermarking methods one for Java bytecode and another for native assembly code. Watermarking Java bytecode is done through first splitting a numeric watermark into multiple values so that a subset can recover the watermark and then inserting each of these numeric values in the program using false predicates. The problem with this scheme is that the attacker can identify the variables whose values are always fixed for different input and can eliminate them. Native code watermarking is done through the use of branch function calls. These are calls which from address A, but return control to some other address A. The encoding used is to embed watermark bit 0, $A < A$, and to embed watermark bit 1, $A > A$. The problem identified by the authors is that the attacker can detect functions modifying their return address, and the solution suggested is to use helper functions (which are intermediate functions) so that a chain is formed that ultimately returns control to A'. As an example $A \rightarrow f_1 \rightarrow f_2 \rightarrow f_3 \rightarrow \ldots \rightarrow f_n \rightarrow A'$. Function f_n effectively modifies the return address $A \rightarrow A'$, which is equivalent to $A \rightarrow f_n \rightarrow A'$. Hence the solution is not really effective. Furthermore, the scheme requires each branch call to be preceded by an unconditional branch statement to the instruction after the branch call so as to bypass it. An automated tool can detect such a sequence (unconditional branch

call followed by another branch call) and upon casual analysis, the attacker can destroy the watermark. [88] by Thomborson et al. again uses PPCTs and repetitive or redundant PPCTs to embed watermarks.

In the field of watermarking through register allocation, Qu and Potkonjak have proposed several watermarking models to embed watermarks in software using graph coloring [75, 76, 79, 78, 77, 95, 57]. The *edge-adding* watermarking technique from [75, 76] is implemented in [70].

For a graph $G = (V, E)$ where V is the set of vertices and E the set of edges between these vertices. The graph can be colored in the following two ways.

1. *Vertex coloring*: If $(v_1, v_2) \in E$, v_1 and v_2 cannot have the same color. That is, vertices having an edge between them cannot have the same color.

2. *Edge coloring*: If $e_1 = (v_1, v_2)$ and $e_2 = (v_2, v_3)$ or $e_2 = (v_1, v_3)$, e_1 and e_2 cannot have the same color. That is, edges that have a common vertex or edges that are adjacent to each other cannot have the same color.

Definition 3.3.1 *Given a graph $G = (V, E)$, a set of 3 vertices $\{v_1, v_2, v_3\}$ is considered a triple and represented as $(v_1, v_2, v_3) \in \mathcal{T}$ if*

1. $v_1, v_2, v_3 \in V$ *, and*

2. $(v_1, v_2), (v_1, v_3), (v_2, v_3) \notin E$

If vertex v_i does not belong to any triple, we represent this as $v_i \notin \mathcal{T}$.

Definition 3.3.2 *Given a graph $G = (V, E)$, a set of 3 vertices $\{v_1, v_2, v_3\}$ is considered a colored triple and represented as $(v_1, v_2, v_3) \in \mathcal{CT}$ if*

1. $v_1, v_2, v_3 \in V$ *, and*

2. $(v_1, v_2), (v_1, v_3), (v_2, v_3) \notin E$*, and*

3. v_1, v_2, v_3 *are all colored the same color*

We also represent as $v_i \notin \mathcal{CT}$ if vertex v_i does not belong to any colored triple.

We use the vertex coloring method for our purpose. The color of a vertex v is represented as $c(v)$. A software can be viewed as a graph with variables as vertices and an edge $E = (v_1, v_2)$ in the graph indicates that the variables v_1 and v_2 are live

concurrently and cannot be assigned the same register and thereby cannot have the same color. Thus, based on the watermark bits, we introduce edges into the graph and change the graph coloring. The major disadvantage of these schemes however is that they are non-blind and require unmarked software copy as well to compare and detect/ extract watermark.

The original QP embedding and extraction algorithms are provided in Algorithm 11 and Algorithm 12 respectively. Implementation of the QP algorithm is performed as follows,

1. Track registers used for storing different variables during the programs execution

2. Each variable is represented as a vertex in a graph

3. If two variables are stored in two different registers, they are connected otherwise not.

4. Using QP algorithm, disconnected vertices are connected depending on watermark bits and physically realized by changing registers that store these variables.

Input : Graph $G = (\{V\}, \{E\})$, Watermark $W = \{w_1, w_2, \ldots, w_n\}$
Output: Watermarked Graph $G' = (\{V\}, \{E'\})$
x=1;
while $i <= n$ **do**
  if *there are two vertices* v_{x_1}, v_{x_2} *nearest to* v_x*, not connected to* v_x **then**
    if $w_i == 0$ **then**
      Add (v_x, v_{x_1}) to E;
    else
      Add (v_x, v_{x_2}) to E;
    end
    $i = i + 1$;
  end
  $x = x + 1$;
end
Return $G' = (V, E')$;

Algorithm 11: QP watermark insertion [75, 76]

Input : Graph $G = (\{V\}, \{E\})$, Watermarked Graph $G' = (\{V\}, \{E'\})$
Output: Watermark detection status
x=1;
while $i <= n$ **do**
  if *Can find vertices* v_{x_1}, v_{x_2} *nearest to* v_x *that are not connected to* v_x **then**
    if $c(v'_x) \neq c(v'_{x_1})$ **then**
      $W' = W' \| 0$;
      Add (v_x, v_{x_1}) to E;
    else
      $W' = W' \| 1$;
      Add (v_x, v_{x_2}) to E;
    end
    $i = i + 1$;
  end
  $x = x + 1$;
end
if *G==G'* **then**
  Watermark detected;
else
  Watermark not detected;
end
Return W';

Algorithm 12: QP watermark extraction [75, 76]

(a) Watermark 010

(b) Watermark 111

Figure 3.9: Watermarks 010 and 111 resulting in the same watermarked graph

The deficiency in the original QP algorithm is that it sometimes maps more than one watermarks to the same resultant graph. Hence, in these situations, detection would always return one particular watermark irrespective of which watermark is actually contained in the watermarked graph. An example is provided in Figure 3.9. This example illustrates how watermarks 010 and 111 both result in the same watermarked graphs thereby making it impossible to deterministically identifying the actual watermark inserted. Figure 3.9 a) shows the steps in embedding watermark 000 while Figure 3.9 b) illustrates watermark 111 being embedded.

The problem of a single watermarked copy corresponding to multiple watermarks is addressed in [70] and a solution to overcome this is proposed. In [70], the message is embedded as follows.

Given a graph $G(V, E)$, and a watermark $W = \{w_1, w_2, \ldots, w_n\}$, the set of vertices V is sorted to $\{v_1, v_2, \ldots, v_m\}$. For each vertex v_i, find two *nearest* vertices v_{i_1} and v_{i_2} that are not connected to v_i , where *nearest* means $i_2 > i_1 > i \pmod n$, the edges $(v_i, v_{i_1}), (v_i, v_{i_2}) \notin E$ and $(v_i, v_j) \in E$ for all $i < j < i_1, i_1 < j < i_2$. If $w_i = 0$, add edge (v_i, v_{i_1}), else add edge (v_i, v_{i_2}). The edges that are added represent fake interferences and force a new coloring of the graph. The embedding process is provided in Algorithm 13.
The watermark is recognized as follows.

Input : Graph $G = (\{V\}, \{E\})$, Watermark $W = \{w_1, w_2, \ldots, w_n\}$
Output: Watermarked Graph $G' = (\{V\}, \{E'\})$
x=1;
while $i <= n$ **do**
  if $v_x \notin \mathcal{T}$ **then**
    If possible, find vertices v_{x_1}, v_{x_2} nearest to v_x that are not connected to v_x;
    if $c(v_{x_1}) == c(v_{x_2}) == c(v_x)$ *&* $v_{x_1} \notin \mathcal{T}$ *&* $v_{x_2} \notin \mathcal{T}$ **then**
      if $w_i == 0$ **then**
        Add (v_x, v_{x_1}) to E;
      else
        Add (v_x, v_{x_2}) to E;
      end
      $i = i + 1$;
    end
  end
  $x = x + 1$;
end
Return $G' = (V, E')$;

Algorithm 13: QPS watermark insertion[70]

Given a graph $G(V, E)$, for each pair of vertices $(v_i, v_j), j > i \pmod n$, which are not connected by an edge and are different colors, one bit of the message can be obtained. The bit extraction is done by examining how many vertices occur between v_i and v_j which are not connected to v_i. The detection process is provided in Algorithm 14.

A further improvement to the original QP algorithm is proposed in the form of QPI algorithm by Zhu and Thomborson in [106, 107]. The main advantage of this work is that the watermark is *extractable* and not merely *recognizable/detectable*. The definition of the the two candidate vertices v_{i_1} and v_{i_2} for vertex v_i is modified by introducing the ordering constraint to be non-cyclic rather than the previous notion of cyclic ordering in [75, 76]. As an example, let there be six vertices $v_1, v_2, \ldots, v_6$. While trying to find a candidate vertex to be connected to v_5, vertices v_1, v_6 are selected. The algorithm in [75] will take $v_6 > v_5$ and $v_1 > v_6$ since it goes in a cyclic manner but the new algorithm [106, 107] takes $v_1 < v_5$ and $v_6 > v_5$. With the modified scheme, two watermarks do not map to the same watermarked relation. Following are the drawbacks of the scheme that need attention.

- The watermarking scheme is non blind since it requires the original graph to

Input : Graph $G = (\{V\}, \{E\})$, Watermarked Graph $G' = (\{V\}, \{E'\})$
Output: Watermark detection status
x=1;
while $i <= n$ **do**
  if $v_x \notin \mathcal{T}$ **then**
    If possible, find vertices v_{x_1}, v_{x_2} nearest to v_x that are not connected to v_x;
    if $c(v_{x_1}) == c(v_{x_2}) == c(v_x)$ *&* $v_{x_1} \notin \mathcal{T}$ *&* $v_{x_2} \notin \mathcal{T}$ **then**
      if $c(v'_x) \neq c(v'_{x_1})$ **then**
        $W' = W' \| 0$;
        Add (v_x, v_{x_1}) to E;
      else
        $W' = W' \| 1$;
        Add (v_x, v_{x_2}) to E;
      end
      $i = i + 1$;
    end
  end
  $x = x + 1$;
end
if *G==G'* **then**
  Watermark detected;
else
  Watermark not detected;
end
Return detection status;

Algorithm 14: QPS watermark extraction [70]

extract the watermark.

- The work does not address security against secondary watermarking.

A straightforward yet effective technique to insert watermark is presented in [72], although with an enormous increase in software size. If watermark bit=0, all sections of a code block executed by a single thread. If watermark bit=1, different sections of a code block executed by different threads.

Building up on the scheme proposed by Monden et al. [69], a watermarking scheme that inserts explicit watermark instructions in dummy methods is proposed in [39]. The new scheme sets the access permission for all the methods and members public, thereby obfuscating the classfiles. The very object oriented technology principles are compromised since encapsulation and data abstraction conditions are violated. Also any class can modify data of any class, which is a critical problem.

Pieprzyk describe a watermarking model [74] that embeds digital signature in the software and verifies the software for the following purposes,

1. *Integrity*
2. *Authority*
3. *Fingerprint*

For a given instruction, there can be n variations that have the same semantic effect. If there are m such instructions, then total combinations are $n * m$. An enumerated list of such combinations is created and combination suitable to insert a particular digital signature is chosen (consisting of author, owner, buyer, seller information). Jarek et al. [51] use similar principle of possible combinations being used for insertion of digital signature, but on classes instead of instructions.

3.3.4 Summary

Table 3.2 provides a summary of comparison between various proposed software watermarking techniques. According to our analysis, branch-based watermarking schemes are more secure than other categories because they exploit the inherent redundancies in softwares. Obfuscation based watermarking appears to be most

fragile since it embeds the watermark in the encoding of classes and instructions. Obfuscation based software watermarking schemes can be compared with format-based text watermarking schemes since both embed watermark in the format and not the data itself.

Attack	[90]	[29]	[28]	[27]	[26]	[88]	[70]	[72]	[39]	[74]	[51]
Object renaming	Secure	Secure	Secure	Secure	Secure	Secure	Insecure	Secure	Secure	Secure	<Secure>
Decompile and Recompile	Secure	Secure	Secure	Secure	Secure	Secure	Insecure	<Secure>	Secure	Secure	Secure
Code Re-ordering	<Secure>	<Secure>	<Secure>	Insecure	<Secure>	Secure	Insecure	Insecure	Insecure	Insecure	Secure
Adding no-effect code	<Secure>	<Secure>	<Secure>	Secure	<Secure>	Secure	Insecure	Insecure	Insecure	Insecure	Secure
Synonymous Object	Secure	Secure	Secure	Secure	Secure	Secure	Secure	Secure	Insecure	Insecure	<Secure>
Checking unreachable code	Secure	Secure	Secure	Secure	Insecure	<Secure>	Secure	Secure	Secure	Secure	Secure
Checking dead code	Secure	Secure	Secure	Secure	Insecure	<Secure>	Secure	Secure	Secure	Secure	Secure
Overhead in time, size	High	High	High	High	Low	High	Low	High	Low	Low	Low

Table 3.2: Comparative study of software watermarking schemes

(<Secure> implies partial security)

3.4 Database Watermarking

Database watermarking is a relatively new comer with a growing interest and increased research activity in the last five years. A typical scenario that requires database watermarking is when a company $\mathcal{C}$ provides confidential customer data to an external organization $\mathcal{ORG}$ (eg. call center). To ensure that $\mathcal{ORG}$ does not exploit the information and doesn't sell it, $\mathcal{C}$ embeds its watermark in the database relation. Another application is in web services, where data provider $\mathcal{D}$ makes a database relation available online for remote queries. An attacker may try to steal the relation using multiple *intelligent* queries. To prevent this, $\mathcal{D}$ watermarks the databases before making them available online using a blind or non-blind watermarking scheme. In a blind watermarking model, only the watermarked media and a secret key are required to detect/extract watermark whereas in a non-blind watermarking scheme, the unmarked multimedia object is also required in addition to the watermarked copy and secret key. This creates a situation where the unmarked object needs to be stored at a secondary secure location.

This section outlines relevant database watermarking schemes, primarily Agrawal and Kiernan's model proposed in [11, 12], which serves as the basis on many other database watermarking schemes. A numeric set watermarking scheme [85] is discussed here before proceeding to the central discussion about database watermarking.

A watermarking scheme for numeric data set is proposed in [85]. The problem can be stated as embedding a bitstream watermark $W = \{w_1, \ldots, w_m\}$ in a data set $\mathcal{S} = \{s_1, \ldots, s_n\} \subset \mathcal{R}$ and creating a watermarked version $\mathcal{V} = \{v_1, \ldots, v_n\} \subset \mathcal{R}$ under *usability* condition (to ensure that the resulting data is useful from the user's perspective). This can be achieved by limiting the distortion for each data set within a range $[g_{min}, g_{max}]$. Various metrics can be used to measure usability such as mean squared error such that the following holds.

$$(s_i - v_i)^2 < g_i \quad \forall i = 1, 2, \ldots, n \quad g_i \in \mathcal{R} \tag{3.1}$$

$$\sum (s_i - v_i)^2 < g_{max} \qquad g_{max} \in \mathcal{R} \tag{3.2}$$

During the insertion process, the numeric set S is split into buckets S_is using Equation 3.3. In the actual implementation, a one-way, secretly keyed, crypto-

graphic hash of the set of most significant bits (MSB) of the normalized version of the items is used to achieve this [85]. We embed a single bit in each subset using an encoding convention and check for data usability bounds. If usability bounds are exceeded, a different encoding parameter variations is tried, and if still there is no success, we try to mark the subset as invalid, and if still there is no success, we ignore the current set. This leaves an invalid watermark bit encoded in the data that will be corrected by majority voting at extraction time. The encoding convention is determined by a confidence factor c and confidence violators hysteresis v_0, v_1 that are input to the insertion algorithm. The watermark is modeled by the percentage of the "confidence violators" present in data subsets for given c, v_0, v_1. The insertion process is provided in Algorithm 15 and the detection process works similarly in terms of dividing the numeric set into subsets and determining whether the bit extracted is valid or not.

$$S_i = \{s_j \in S | (k_i)_{bit_j} = 1\}, i = 1, \ldots, m \tag{3.3}$$

It is quite evident from [85] that *usability* plays a big role in determining the overall quality of a watermarking scheme, especially when database relations are the objects. Gross-Amblard proposed one of the few watermarking models that address the usability issue in detail [40]. *Usability* is measured in terms of query results. If the results of a given set of queries are preserved for a watermarked relation, the watermarking is said to be query-preserving, and is desirable. The notions of *local* and *global distortions* are presented in [40] which achieve the property of query-preservation. The user can execute queries $q_1, q_2, \ldots, q_k$ on the database relations. The proposed watermarking scheme respects the following conditions.

1. the watermarking scheme transforms the database relation into several versions, and results in a small distortion on the query results $q_1(u'), q_2(u'), \ldots, q_k(u')$ and the owner acts as any user u'.

2. the scheme can prove ownership based on answers to the above k queries only.

We are going to use Table 3.3 and Table 3.4 to explain the concepts from [40]. The *tuple weight* $W(t_j)$ is the numerical value associated with that tuple. In case of multiple numerical values in a tuple, the summarizing value or the most valuable value is selected for computing weight. For example, the weight of a tuple in Table 3.4 is the cost of the meals.

Input : Numeric set $S = \{s_1, \ldots, s_n\} \subset \mathbb{R}$, Watermark $W = \{w_1, \ldots, w_m\} \subset \{0, 1\}$, Key $\mathbb{K} = \{k_1, \ldots, km\}$, confidence factor c, confidence violators hysteresis v_0, v_1
Output: Watermarked numeric set S_w
$index(s_i) = H(k_s, MSB(NORM(s_i)), k_s)$;
Divide S into subsets based on $index(s_i)$;
$d = 1$;
while *watermark to be embedded* **do**
  $avg(S_d) = \frac{\sum x_j}{|S_d|} \forall x_j \in S_d$;
  $\delta(S_d) = \sqrt{\frac{\sum (avg(S_d)x_j)^2}{|S_d|}} \forall x_j \in S_d$;
  $f_c = |x_j| x_j > avg(S_d) + c \times \delta(S_d)\ v_c(S_d) = |f_c|$;
  if $v_c(S_d) > v_1 \times |S_d|$ **then**
    $mark = 1$;
  else
    if $v_c(S_d) < v_0 \times |S_d|$ **then**
      $mark = 0$;
    else
      $mark = invalid$;
    end
  end
  $d = d + 1$;
end

Algorithm 15: Watermark insertion in numeric set

Meal Type	Code
Breakfast	B1
Breakfast	B2
Breakfast	B3
Lunch	L1
Lunch	L1
Dinner	D1
Dinner	D2
Dinner	D3

Table 3.3: Meal table

Code	Item 1	Item 2	Item 3	Item 4	Item 5	Cost($)
B1	Omelette	Cereal	Milk	Apple Juice	Coffee	10.50
B2	Omelette	Fruits	Milk	Apple Juice	Coffee	11.50
B3	Cereal	Fruits	Milk	Orange Juice	Tea	9.60
L1	Sandwich	Fries	Juice	-	-	8.60
L2	Noodles	Wings	Juice	-	-	9.60
D1	Steak	Corn	Salad	Cake	Wine	20.90
D2	Steak	Fries	Salad	Pie	Beer	18.90
D3	Burger	Fries	Salad	Cake	Beer	18.90

Table 3.4: Combination table

A *Weight function* f for any attribute in the table is defined in the papers as sum of the weights of tuples in a query result. Function f is used to control the distortions on query results (the operation $\sum$ in the weight function can be modified to $mean, min, max$ without effecting the overall quality of the watermarking scheme). The weight function and an example is provided below,

$$f(A_i) = \sum W(t_j : A_i \in t_j) \tag{3.4}$$

$$f(Breakfast) = Weight(B1) + Weight(B2) + Weight(B3) = 22.10$$
$$f(Lunch) = Weight(L1) + Weight(L2) = 18.20$$
$$f(Dinner) = Weight(D1) + Weight(D2) + Weight(D3) = 58.70$$

Local distortion caused by modification on a database is defined as $|W(t_j) - W'(t_j)|$ and *global distortion* is defined as $|f(A_i) - f'(A_i)|$. Given a constant c, a watermarking scheme is said to respect *c-local distortion* assumption if $|W(t_j) - W'(t_j)| <= c$. Furthermore, given a constant d, a watermarking scheme is said to respect *d-global distortion* assumption if $|f(A_i) - f'(A_i)| <= d$.

For example, let Table 3.5 and Table 3.6 be two different variations of Table 3.4. For $c, d = 1$, combination 1, respects *1-local distortion* but not *d-global distortion* because $|f(Dinner) - f(Dinner')| = 2$ and also $f(Lunch) - f(Lunch') = 2$. Combination 2 however respects both *1-local distortion* and *d-global distortion* because none of the tuple weights are modified beyond 1.00 and $f(Breakfast) - f(Breakfast') = 0.40$, $f(Lunch) - f(Lunch') = 0.40$ and $|f(Dinner) - f(Dinner')| = 0$.

Code	Item 1	Item 2	Item 3	Item 4	Item 5	Cost($)
B1	Omelette	Cereal	Milk	Apple Juice	Coffee	9.90
B2	Omelette	Fruits	Milk	Apple Juice	Coffee	10.50
B3	Cereal	Fruits	Milk	Orange Juice	Tea	8.60
L1	Sandwich	Fries	Juice	-	-	9.60
L2	Noodles	Wings	Juice	-	-	10.60
D1	Steak	Corn	Salad	Cake	Wine	19.90
D2	Steak	Fries	Salad	Pie	Beer	17.90
D3	Burger	Fries	Salad	Cake	Beer	18.90

Table 3.5: Version 1 of combination table

Code	Item 1	Item 2	Item 3	Item 4	Item 5	Cost($)
B1	Omelette	Cereal	Milk	Apple Juice	Coffee	10.90
B2	Omelette	Fruits	Milk	Apple Juice	Coffee	11.20
B3	Cereal	Fruits	Milk	Orange Juice	Tea	9.90
L1	Sandwich	Fries	Juice	-	-	8.70
L2	Noodles	Wings	Juice	-	-	9.90
D1	Steak	Corn	Salad	Cake	Wine	20.00
D2	Steak	Fries	Salad	Pie	Beer	19.80
D3	Burger	Fries	Salad	Cake	Beer	18.90

Table 3.6: Version 2 of combination table

Active weighted elements: ($\mathcal{W}$) The elements that are modified in a particular version of watermarking activity are included in *active weighted elements*. In the given example, $\{B1, B2, B3, L1, L2, D1, D2\}$ are active weighted elements while $D3$ is not an active weighted element since it is not modified in versions combination 1 or combination 2. As a result, we only have $\mathcal{W}$ elements to watermark.

If we can find 2^l distinct distortions respecting both *c-local distortion* and *d-global distortion* for predefined c, d, then we can embed a distinct $l-bit$ watermark in each variation of the relation and distribute it to a different server, thereby being able to identify any of the 2^l malicious servers.

The paper focuses on limiting global distortion and restricts the value of c to 1. The marking algorithm $\mathcal{M}$ inputs relation set and global distortion limit R, d respectively and outputs a distorted relation set R' containing a $l - bit$ watermark such that $\Pr(R'$ respects $d-$global distortion$) > 0.75$.

Let R'' be a d'-global distortion of R. The detection algorithm $\mathcal{D}$ inputs all possible query answers from suspect server using R'' and $\Pr(\mathcal{D}$ outputs m$) > 1 - \delta$ where δ is the failure probability of the detector $\mathcal{D}$.

Amongst the LSB-modifying watermarking schemes, that are usually susceptible to random bit flipping attacks, we discuss here a second-LSB based database watermarking proposed in [97]. It inserts a watermark in a relational database table by pseudo-randomly grouping attributes and changing the least significant digit of the attributes of the same group to the same digit if the bit to be inserted is 1, otherwise introducing no change. Consider the original relation in Table 3.7 as an example. Let the underlined attributes belong to the same group and the watermark bit is 1, then a random value $c \in \{0 \ldots 9\}$ (say 8) is generated, and the least significant digit of these attributes is changed to c. In this case, the resulting relation is Table 3.8. In the case that watermark bit is 0, then the watermarked relation is given in Table 3.9.

More is the number of the 1-bit in the watermark, more are the changes introduced by the embedding algorithm. An all-0-bit watermark does not introduce any change in the document. Hence the modification in the database relation is proportional to the number of 1's in the watermark.

If all attributes of the relation having the same least significant digit (LSD) are grouped together, then one or more of these groups also contain the group G embedding a watermark bit 1. If the LSD of attributes in these groups are modified

primary key	attribute
January	**53**
February	63
March	**41**
April	56
May	89
June	39
July	**77**
August	102
September	65
October	59
November	91
December	**76**

Table 3.7: Original Table

primary key	attribute
January	**58**
February	63
March	**48**
April	56
May	89
June	39
July	**78**
August	102
September	65
October	59
November	91
December	**78**

Table 3.8: Watermarked with bit 1

primary key	attribute
January	**53**
February	63
March	**41**
April	56
May	89
June	39
July	**77**
August	102
September	65
October	59
November	91
December	**76**

Table 3.9: Watermarked with bit 0

uniformly to $0, 1, \ldots, 9, 0, 1, 2 \ldots$, then the distribution of G is also modified accordingly thereby destroying the watermark bit with a high probability. Algorithm 16 formally describes the attack. This attack focuses on randomization and therefore only watermark bits 1 will be attacked by this method. From the embedder's perspective, an apparent security measure against this kind of attack would be to have very few 1's in the embedded watermark but this would lead to a high probability of false positives. In order to limit the false positive probability below a pre-determined threshold, there must be a lower bound on the ratio 1's to 0's in the watermark. We implemented our attack in C++ and tested the first prototype with single attribute relations and then with multi-attribute relations. We have achieved 20-25% attack success with 40-45% modification. The source code for the attack can be found in the appendix. Summarization of results is given in Figure 3.10.

3.4.1 Agrawal-Kiernan Database Watermarking Scheme

Agrawal and Kiernan present a watermarking scheme that modifies LSBs of pseudo-randomly selected numerical attributes in a relational database [11, 12]. Following are the parameters to the watermarking algorithm.

- Database relation R.
- Number of tuples, η, in R.

Input : Watermarked relation R
Output: Attacked relation R'

```
for k =1 to 10 in steps of 1 do
    counter_k = random (mod 1)0;
end
for tuple r ∈ R do
    for attributes A_i ∈ r do
        A_i = (A_i/10) * 10 + counter_{A_i};
        counter_{A_i} = (counter_{A_i} + 1) (mod 1)0;
    end
end
```

Algorithm 16: Uniform distribution attack

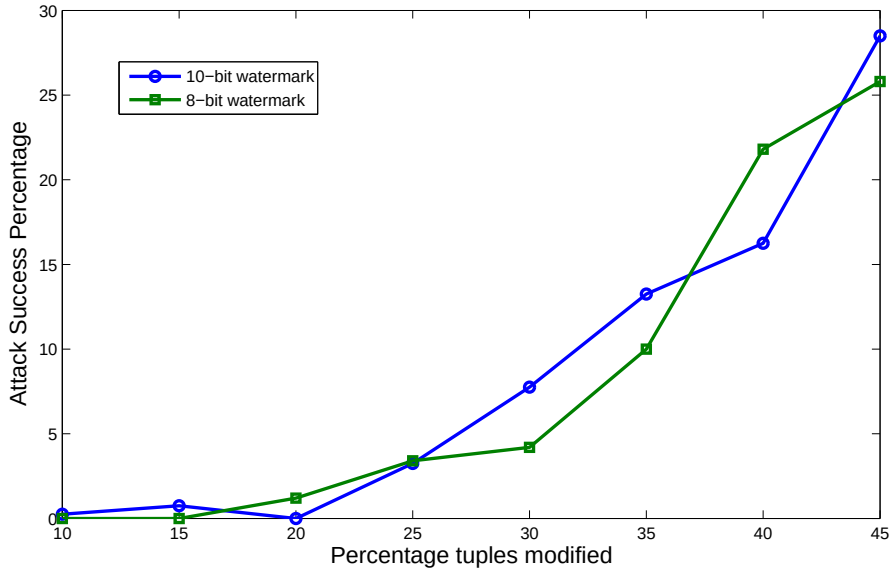

Figure 3.10: Launching an attack on second-LSB based watermarking

- Number of attributes, υ, in R available for watermarking.
- Number of LSBs, ξ, available for watermarking.
- Fraction of tuples, $1/\gamma$, to be watermarked.
- Significance level, α, for watermark detection.
- Minimum number of correctly marked attributes, τ, for successful detection.

The basic operation in the watermark inserting algorithm is a message authenticated code given in Equation 3.5, where $\|$ represents concatenation and $\mathcal{H}$ is a one-way hash function.

$$\mathcal{F}(r.P) = \mathcal{H}(\mathcal{K}\|\mathcal{H}(\mathcal{K}\|(r.P)) \tag{3.5}$$

The watermark insertion algorithm is given in Algorithm 17. The algorithm selects pseudo-randomly the tuples, the attributes and the bit to be watermarked. The size of the mark *inserted* is γ bits. Since the embedding procedure involves a sequential and independent watermark bit embedding in tuples, the detection process is essentially re-visiting the tuples (in any given order, since they are independently marked) and trying to detect or validate the watermark bit in that tuple.

The detection algorithm is provided in Algorithm 18. The detection algorithm checks whether the detected bit matches the actual bit or not. Identification of the correct tuple, attribute, and bit position relies on $\mathcal{F}(r.P)$ where $r.P$ is the primary key of the r^{th} tuple. In the algorithm, τ is the minimum number of bits that need to be detected correctly such that the probability of falsely detecting τ out of *totalcount* bits is less than α.

The attacks considered by the authors include the following.

A1: *Bit attack*: Attacker updating some bits in numerical attributes.

A2: *Randomization attack*: Attacker assigns random values to some bits.

A3: *Rounding attack*: Attacker rounds off certain number of bits.

A4: *Translation attack*: Attacker transforms numerical values to suit another unit of measurement.

A5: *Subset attack*: Attacker extracts a subset of tuples or attributes.

Input : relation R, private key K, fraction $\frac{1}{\gamma}$, LSB usage ξ, Markable attributes $\{A_1, A_2, \ldots, A_\upsilon\}$, primary key P
Output: Watermarked relation R_w

forall *tuples* $r \in R$ **do**
  if $\mathcal{F}(r.P) \pmod{\gamma} = 0$ **then**
    Mark attribute $i = \mathcal{F}(r.P) \pmod{\upsilon}$;
    Mark bit $j = \mathcal{F}(r.P) \pmod{\xi}$;
    $r.A_i^j = \mathcal{H}(\mathcal{K} \| r.P) \pmod 2$;
  end
end

Algorithm 17: Watermark insertion [11]

Input : relation R, private key K, primary key P, fraction $\frac{1}{\gamma}$, LSB usage ξ
Output: Watermark Status $\in \{true, false\}$

$matchcount = 0$;
$totalcount = 0$;
forall *tuples* $r \in R$ **do**
  if $\mathcal{F}(r.P) \pmod{\gamma} = 0$ **then**
    Marked attribute $i = \mathcal{F}(r.P) \pmod{\upsilon}$;
    Marked bit $j = \mathcal{F}(r.P) \pmod{\xi}$;
    if $\mathcal{H}(\mathcal{K} \| pk) \pmod 2 = r.A_i^j$ **then**
      $matchcount = matchcount + 1$;
    end
    $totalcount = totalcount + 1$;
  end
end
$\tau = \min(\theta) : \mathrm{B}(\theta, \mathrm{totalcount}, 1/2) < \alpha$;
if $matchcount \geq \tau$ **then**
  return true;
else
  return false;
end

Algorithm 18: Watermark detection [11]

A6: *Mix and match attack*: Attacker applies **A4** on multiple relations and create a new merged relation.

A7: *Additive attack*: Attacker watermarks an already watermarked relation with his/her watermark.

A8: *Invertibility attack*: If the detection algorithm returns *detected* for a random key chosen by the attacker, then (s)he can claim ownership of that relation.

During detection, each tuple is individually checked for containing the watermark bit. If an attacker inserts new tuples, it does not distort the watermark arrangement. There is a probability of $\frac{1}{\gamma}$ detection algorithm selecting the tuple added by the attacker. Even if the added tuple is selected, the bit will be identified as correctly marked or incorrectly marked with equal probabilities. Thus, an attacker may be successful in causing detection of one incorrect bit with a probability of $\frac{1}{2*\gamma}$ but at the same time might actually contribute to the successful detection of the watermark while trying to destroy the watermark with the same probability.

The watermark detection status depends on the ratio of number of the correctly detected bits, k, to the number of the total tuples identified as marked, n. If an attacker deletes some tuples, both these values are reduced and thus there is very little effect on the overall ratio $\frac{k}{n}$. This provides security against subtractive attacks.

Bit flipping attacks (as a family) are also ineffective with high probability since the identification of correct tuples, attributes and LSBs is dependent on the keyed-MAC.

The watermarking scheme appears to be satisfactorily secure against distortive attacks. However, there is no discussion on query preservation or data usability and the scheme suffers from certain drawbacks in terms of database usability and correctness. The following example from Table 3.10 explains situations in which lack of query preservation might be a concern. The primary key in Table 3.10 is *Currency Code*. Let the watermark insertion algorithm modify the relation to Table 3.11.

In such a situation, the following two queries will return incorrect results as in the following cases.

Select Country ***from*** ForEx ***where*** Selling rate < 130. (returns *Australia* only in unmarked relation and returns *Australia*, *New Zealand* in watermarked re-

Currency Code=P	Nation	Buying rate	Selling rate
AUD	Australian	133	125
INR	Indian	4500	4300
THB	Thailand	3740	3510
SLR	Sri Lanka	4430	4210
NZD	New Zealand	151	134

Table 3.10: Foreign exchange rates

Currency Code=P	Nation	Buying rate	Selling rate
AUD	Australian	133	125
INR	Indian	4500	4300
THB	Thailand	3740	3510
SLR	Sri Lanka	4530	4310
NZD	New Zealand	151	124

Table 3.11: Foreign exchange rates (watermarked)

lation).

Select Currency Code ***from*** ForEx ***where*** Buying rate is maximum. (returns *India* in unmarked relation and returns *Sri Lanka* in watermarked relation).

This shows that the watermarking scheme does not preserve queries, and this is a major drawback. A possible solution to this problem is that attributes are marked at most by $\phi_i = minimum(|r_j.A_i - r_k.A_i|), \forall i, j, k$. The upper limit on modification of i^{th} attribute can be derived from ϕ_i. This would guarantee query-preservation for the current relation but not for any future updates. But no watermarking scheme can guarantee query-preservation for unrestricted expansion of the relation. An informal proof this claim is given here. Let the smallest difference between values of attribute A_i be ϕ_i and the minimum value of A_i be $r_s.A_i$. If $r_s.A_i$ is watermarked by a query-preserving watermarking scheme, it's new value can be in the range of $r_s.A_i \pm (\phi_i - \delta)$. Without loss of generality, let it be $r_s.A_i + \phi_i - \delta$. Inserting a tuple r_t with $r_t.A - i = r_s.A_i - \phi_i + \delta$ (which is the old value of $r_s.A_i$) in the relation would return r_t as an answer to the query ***Select*** r ***from*** R ***where*** A_i is minimum. This is despite the fact that the actual value of $r_s.A_i$

Currency Code=P	**Nation**	**Buying rate**	**Selling rate**
aud	Australian	133	125
inr	Indian	4500	4300
thb	Thailand	3740	3510
slr	Sri Lanka	4530	4310
nzd	New Zealand	151	124

Table 3.12: Table with modified primary key

is minimum. Thus, the best result we can achieve is to ensure that the current relation instance is watermarked in a query-preserved format while understanding that future modifications on the relations may distort query results.

Secondly, the scheme uses a fixed precision parameter ξ irrespective of the attribute it watermarks. This is a drawback since some attributes are more *sensitive* than others. For example, *Quantity* of potatoes exported from *Australia* can accept a modification of a few hundreds or even thousands but *buying rate* and *selling rate* can tolerate a modification of only a few cents. It is therefore recommended that the number of LSBs considered for watermarking should be a function of pre-defined *sensitivity* of the attributes. The number of usable LSBs would be $\{\xi_1, \xi_2, \ldots, \xi_\upsilon\}$, where υ attributes are available for watermarking.

Presence of a primary key is a pre-requisite for watermarking. Without it, the algorithm cannot identify the tuples to be marked and detected correctly. Although this is not a completely unreasonable or impractical requirement, it does limit the capability of the watermarking scheme in theoretical terms. What is more important is that the tuples, attributes, and LSBs to be watermarked are determined using a MAC on the primary key's value, $\mathcal{F}(r.P)$. If the primary key values are scaled or modified by a constant value (for example, by prefixing or suffixing with a constant bitstring or changing cases), the binary value would differ, which would result in failure during watermark detection as in Table 3.12.

In conclusion, the list below shows the major problems with the existing scheme of Agrawal and Kiernan.

1. The number of LSBs that can be used for watermark insertion are not dependent on the sensitivity of the attribute but is fixed for all attributes. This can result in the relation being unusable from a user's perspective.

2. Distance between different values of the same attributes is not taken into consideration which may alter query results.

3. The scheme relies on the primary key to identify the tuple, attribute and LSB to be marked. The attacker can perform format-based modifications on the primary key, thereby still retaining the usability of the primary key while changing its bit-value. As a simple example, the attacker can change the primary key from lower case to upper case and vice versa. The function $\mathcal{F}(r.P')$ would therefore be different from $\mathcal{F}(r.P)$.

4. The scheme does not protect against secondary watermarking. The attacker can choose his/her own parameter list $(pk', \gamma', \upsilon', \xi')$ and insert a new watermark. This watermark will probabilistically destroy the original watermark and deterministically establish the ownership of the attacker over the relation.

   However, if we change the equation $r.A_i^j = \mathcal{H}(\mathcal{K}\|r.P) \pmod 2$ in Algorithm 17 to the following,

   $$r.A_i^j = \mathcal{H}(\mathcal{K}\|pk) \oplus r.A_i^j \pmod 2$$

   where $r.A_i^j$ is the j^{th} bit of the i^{th} attribute in the current tuple, then the watermarking scheme would become reversible. The problem with the proposed solution is that during the new version of detection algorithm, the condition for successful match is that $\mathcal{H}(\mathcal{K}\|pk) \oplus r.A_i^j \pmod 2$ should be the same as j^{th} LSB where $r.A_i^j$ is the old bit at that position. Hence, the existing solution results in a non-blind watermarking scheme, which is a drawback.

   To work around this problem, the bits changed during insertion algorithm $(r.A_i^j)$ can be concatenated and used as a **watermark** W. Hence, the watermark is effectively generated during the insertion algorithm's execution. The new condition to increment *matchcount* would be if $\mathcal{H}(\mathcal{K}\|pk) \oplus w_l \pmod 2$ is same as $r'.A_i^j$, $(1 \leq l \leq |W|)$. These modifications result in the scheme being both reversible *and* blind.

Consider the relation in Table 3.13. Say, $\mathcal{H}(\mathcal{K}\|pk) \oplus r.A_1^2$ is even. Hence, the new bit is 0 and $w_1 = 1$. The watermarked relation is given in Table 3.14.

$$\mathcal{H}(\mathcal{K}\|pk) \oplus w_1 \pmod 2 = r'.A_1^2 = 0 \Rightarrow matchcount = matchcount + 1$$

Currency Code=P	Nation	Buying rate	Selling rate
AUD	Australian	**101*1***	1100
INR	Indian	100010	100101
THB	Thailand	11101	11010
SLR	Sri Lanka	111110	111111
NZD	New Zealand	1101	1110

Table 3.13: Table with binary representation of numerical values

Currency Code=P	Nation	Buying rate	Selling rate
AUD	Australian	**100*1***	1100
INR	Indian	100010	100101
THB	Thailand	11101	11010
SLR	Sri Lanka	111110	111111
NZD	New Zealand	1101	1110

Table 3.14: Watermarked table with binary representation of numerical values

To highlight the strength provided by these modifications, consider the following situation,

Role players: **Owner** *Charles*, **Data Server** *David*, **Judge** *Jones*

Charles claims that a relation R belongs to him and that he watermarked it using $insert(pk, \gamma, \upsilon, \xi)$. He also claims that the version *David* has, is stolen. *David* claims that it's his relation watermarked with $insert(pk', \gamma', \upsilon', \xi')$. *Jones* demands the parameter set P_c, P_d for detecting the watermark from *Charles* and *David* respectively.

In the first sequence, *Jones* executes the detection program with P_c to get watermark detected status S_{11} and reverses the relation to it's original state R_1 and runs the detection program again with P_d to get S_{12}.

In the second sequence, *Jones* executes the detection program, this time, with P_d first to get watermark detected status S_{21} and reverses the relation to it's original state R_2 and runs the detection program again with P_c to get S_{22}. Table 3.15 summarizes the possible outcomes assuming both the parties **HAVE** inserted their watermarks at some point of time.

$S_{11}\|S_{12}$	$S_{21}\|S_{22}$	Outcome
00	11	Belongs to Charles
11	00	Belongs to David

Table 3.15: Owner identification possibilities

1. If *Charles* is the legitimate owner, then *David* has inserted his watermark after *Charles*, the judge would successfully detect $David's$ watermark in R and $Charles'$ watermark in R_1. Hence, $S_{11} = S_{12} = 0$ and $S_{21} = S_{22} = 1$ (row 1).

2. If *David* is the actual owner of the relation, and $Charles'$ claims are false, then *Charles* has inserted his watermark after David (row 2).

The next point to investigate is in what conditions can a pirate (assumed) David destroy the watermark of Charles (assuming there exists one). There are two conditions of this to happen,

1. *Charles' watermark is destroyed inadvertently because of David inserting his watermark*: Since the watermarking is reversible, the original state of the relation can be obtained from detection algorithm on Davids watermarked copy. This will provide the watermarked copy of Charles. Hence, the original *watermark* cannot be accidently destroyed.

2. *David destroys Charles watermark on purpose*: This condition is already discussed in the original paper by Agrawal and Kiernan.

This scenario can be extended to identification of the correct owner when n parties watermark the relation. The condition is that when each user's watermark is correctly detected, the user whose watermark is detected at last is the correct owner. This can be carried out intelligently without using $(n!)$ sequences.

In this section, we have seen that following are the major issues surrounding database watermarking schemes, and these shall be addressed during discussion of our proposed watermarking schemes in Chapters 6, 7.

1. Watermark robustness

2. Data usability

3. Limited distortion

4. Reversibility

5. Resilience against secondary watermarking

3.5 Conclusion

The current chapter describes and analyzes the important works in the field of text, software and database watermarking. It also discusses the generic attacks and attacks specific to watermarked multimedia objects of different types. In particular, we are interested in the following three works.

1. Natural language watermarking [14]: Atallah et. al described a semantics-based natural language watermarking in this work. We improve this scheme by localizing the attack, if any, to the section of the document modified and thereby improve the watermark resilience.

2. Software watermarking [71]: Myles and Jin presented a software watermarking scheme using branch-manipulations but the scheme is vulnerable to debugger-based automated attacks. We present an alternative scheme to survive such automated attacks.

3. Database watermarking [11]: Agrawal and Kiernan proposed a fundamental database watermarking scheme that uses the primary key to identify the values to be watermarked during detection and values from which watermark should be extracted during detection. This scheme has very high security against most attacks except secondary watermarking attacks and the watermark-carrying capacity is sufficiently satisfactory and can be changed using fraction of tuples to be watermarked. We would like to build on this watermarking model to propose a design that is both secure and *reversible* so that secondary watermarking attacks can be defeated and also original relations can be regenerated from watermarked relations.

Chapter 4

Natural Language Watermarking

If we consider the total internet traffic, then a bulk of the data transferred is in form of emails, blogs, web pages, instant messenger conversations, document attachments (written in several languages), memos, and other form of textual information. While most of this data might be publicly available and of little value, some of the information is quite important for governmental and private organizations from a commercial and/or security perspective. These organizations would like to establish their ownership over the documents in case there is a leak of information from within the organization or because of an outside attack. This is the major objective of natural language watermarking.

Natural languages are difficult to watermark because text manipulations are guided by strict rules in terms of grammar, syntax, semantics, context-based selection of a word from a set of synonymous words, etc; while in the case of other media, there is large amount of redundant information to manipulate. For example, human visionary system cannot distinguish between an original image and a watermarked image with the last few LSBs in certain pixels flipped. Similar is the case with audio and video files. But in text documents, grammatical rules need to be preserved while making any changes. There has been significant work done in format-based text watermarking using inter-word and inter-space spacing, justification, alignment, character height and width, etc [18, 19, 24, 25, 54, 67, 98]. The common problem these techniques have is that watermark cannot survive reformatting and reproduction attack as they introduce loss of formatting information in

the document. Alternatively, the attacker can simply re-type the entire document which would be watermark-free.

Synonym substitution watermarking schemes such as [56] are resilient to the above mentioned trivial attacks but not to random synonym substitutions made by the attacker. Even more importantly, words cannot always be replaced by their *exact* synonyms. Hence, the quality of the documents is depreciated by synonym substitution.

4.1 Current Scenario

Recent focus in natural language watermarking has been on syntactic watermarking [14, 100], where language syntax structures are modified to embed watermarks. Notable progress has been made in [14], where watermark bits are embedded in sentences using the following transformations.

1. **Adjunct movement**: Inserting an adjunct (for example, *Usually*, *Generally*, etc) at many of the possible positions in a sentence.
2. **Clefting**: Explicit emphasis on the mandatory subject in the sentence. (for example, the sentence **"We are concerned with** $< subject >$**"** is converted to the sentence **"*It is* $< subject >$ we are concerned with"**)
3. **Passivization**: Changing of voice from active to passive and vice versa. (for example, the sentence **"He led me"** is converted to **"I was led by him"**)
4. **Combination of the above**.

The scheme proposed by Atallah et. al [14] discussed in Section 3.2 has the following drawbacks.

1. introduces a considerable overhead because of parsing each sentence, numbering the nodes and creating a hash for each node.
2. requires a *marker* of the sentence to be added and consequently, reduces the watermark-carrying capacity.
3. becomes vulnerable to multiple sentence transformation attacks.

Table 4.1 summarizes the central ideas of the marking schemes that can be found in the literature.

Modifications made based on watermark bit	**Scheme**
Interword spacing (example - 10-pixels if bit=0; 11-pixels otherwise)	[18, 24, 67, 98]
Interline spacing (example - 10-pixels if bit=0; 11-pixels otherwise)	[18, 67, 98]
Abbreviation and Synonym substitution x (example - *must* if bit=0; *should* otherwise) (example - *a.m.* if bit=0; *A.M.* otherwise)	[56]
Sentence structures (example - *He led me* if bit=0; *I was led by him* otherwise)	[14]

Table 4.1: Natural language and text watermarking methods

4.2 Outline of Proposed Scheme

In the proposed watermarking scheme, the sequence of the paragraphs and sentences used to embed the watermark is permuted. This causes the attack to be limited to a small section and not affecting other sections of the document. So if one paragraph of the document is modified by the attacker, only the watermark bits from that paragraph are affected. Watermark bits are physically embedded by modifying the sentences word counts. Error-correcting codes and majority voting are used to embed watermark bits at multiple locations providing increased security against attacks. The watermark contents are signed using private key of user and publisher which prevents the publisher from framing an innocent user. The watermark bitstream contains a collusion-secure code (described in detail later) to identify colluding users.

4.2.1 Type of Adversary

The attacker is assumed to have the capabilities to do the following.

1. add and/or delete sentences from the document.
2. swap sentences within the same or between different paragraphs.
3. make natural language transformations on sentences.
4. shuffle paragraphs in the document.
5. collude with other users to compare and modify the document.

4.2.2 Mathematical Model and Definitions

Mathematical Model

We represent the watermarking scheme as WS, where

$$WS =< \{P, W_i, K\}, \{\xi, \zeta, \psi\} > \tag{4.1}$$

P is a collection of paragraphs $\{p_1, p_2, \ldots, p_y\}$

p_i is the i^{th} paragraph and contains x_i sentences, i.e. $p_i = \{s_{i1}, s_{i2}, \ldots, s_{ix_i}\}$

d_{ij} is the number of tokens/words in s_{ij}

$W_i = \{w_1, w_2, \ldots, w_n\}$ is the watermark to be inserted where, $\forall i, w_i \in \{0, 1\}$

K is a k-bit secret key

Watermark insertion $\xi : W_i \times P \times K \rightarrow P^{(w)}$, where $P^{(w)}$ is watermarked text

Watermark extraction $\zeta : P^{(w)} \times K \rightarrow W_e$ (extracted watermark)

Watermark verification $\psi : W_e \times W_i \rightarrow \{true/false\}$

Definitions

- $d_i = |s_i|$ gives the number of words in sentence s_i
- $d_i = \{d_{i,1}, d_{i,2}, \ldots, d_{i,k_i}\}$ where $d_{i,j} \in \{0, 1\}$, $k_i = \lceil \log_2 d_i \rceil$ is the binary representation of d_i with b_{i1} as the LSB and so on.
- Watermark $W = \{w_1, w_2, \ldots, w_m\}$ where w_i is the i^{th} bit of the watermark.
- Lexicographically sorted permutations for a set of n elements are $\rho_1^n, \rho_2^n, \ldots, \rho_{n!}^n$. ρ_i^n gives the i^{th} permutation of n elements. $\varrho_{i,j}^n$ gives the value of the j^{th} element in ρ_i^n.

- Majority Voting - $\forall i, a_i \in \{0, 1\}$.

$$majority(a_1, a_2, \ldots, a_n) = \begin{cases} 1 & \text{if } |a_i = 1| > \frac{n}{2} \\ 0 & \text{otherwise} \end{cases}$$

- Text document $P = \{p_1, p_2, \ldots, p_y\} = \left\{\{s_{\alpha_1+1}, \ldots, s_{\alpha_1+x_1}\}, \ldots, \{s_{\alpha_y+1}, \ldots, s_{\alpha_y+x_y}\}\right\}$
  where p_i is the i^{th} text paragraph and s_i is the i^{th} text sentence.

  $p_i = \{s_{\alpha_i+1}, s_{\alpha_i+2}, \ldots, s_{\alpha_i+x_i}\}$

$$\alpha_i = \begin{cases} 0 & \text{if } i = 1 \\ \sum_{j=1}^{i-1} x_j & \text{if } 2 \leq i \leq y \end{cases}$$

- $|p_i|$ defines number of sentences it contains.
- τ = number of paragraphs in which each watermark bit is embedded.

4.3 Proposed Scheme

In order to limit distortions caused by modifications made by the attacker, we permute the sequence of sentences and paragraphs used to embed the watermark. Sentences and paragraphs are not physically permuted but only the sequence in which they will be *picked* to embed the watermark is permuted. Embedding each watermark bit in multiple paragraphs (say μ) results in any $\frac{\mu}{2} + 1$ unmodified bits results in successful recovery of the watermark.

4.3.1 Sequence Permutation

In the current implementation, we have used AES algorithm to generate pseudo-random permutations. With AES, the key size $k \in \{128, 192, 256\}$. However, there are several other methods to generate permutations.

1. The set of paragraph indices $\{1, 2, \ldots, y\}$ is sorted in ascending order of the number of sentences they contain to $G = \{g_1, g_2, \ldots, g_y\}$ such that for $i < j$, we have $|p_{g_i}| \leq |p_{g_j}|$. This helps to counter the paragraph shuffling attack.
2. In binary notation, $\delta = \lceil \log_2 y \rceil$ bits are required to represent index of any given paragraph in a set of y paragraph.

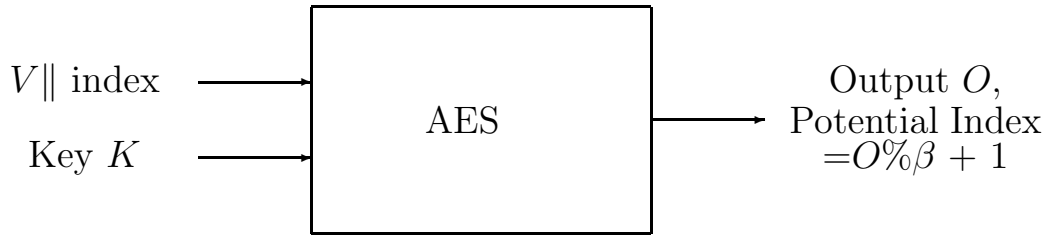

Figure 4.1: Generating a paragraph permutation using AES

3. Vector V is a k-bit vector initialized to the secret V_{iv}.

4. For any index i, input to AES is $V \| i$ and key is K. The first δ bits of encrypted output $\pmod{y}$ gives the paragraphs position θ_i in the new sequence.

5. If V generates a *valid permutation* $(\forall(i,j),\ i \neq j, 1 \leq i \leq y, 1 \leq j \leq y, \theta_i \neq \theta_j)$, final test vector V_f=V, otherwise reject V, repeat step 3,4 with $V = V + 1$.

6. The new paragraph sequence is given by $\{\theta_1, \theta_2, \ldots, \theta_y\}$. This essentially means that p_{θ_i} is used before p_{θ_j} if $i < j$. As an example, if the sequence set is {5, 1, 2, 3, 4} such that $\theta_1 = 5$ and $\theta_2 = 1$, then paragraph 5 is used **before** paragraph 1 in watermark embedding.

7. For $1 \leq i \leq y$, $\rho^{x_i!}_{(\theta_i)^K \pmod{x_i!}}$ is the new sequence of the sentences to be used within the paragraph i. This permutation is generated using Algorithm 19. It generates ${x_i}^{th}$ permutation from a lexicographically sorted set of permutations.

8. The resulting paragraph sequence is $\Theta = \{\theta_1, \theta_2, \ldots, \theta_y\}$ and the sentence sequence is given in Table 4.2

As an illustration, let a document contain 5 paragraphs $\{a, b, c, d, e\}$ with 7, 8, 5, 3, and 6 sentences respectively. Let the new paragraph sequence be {4, 1, 2, 5, 3} and the new sentence sequence be {2, 1, 3} for paragraph d (which is now in first position), {5, 3, 7, 2, 4, 1, 6} for paragraph a, {8, 1, 4, 2, 3, 7, 5, 6} for paragraph b, {3, 4, 6, 2, 5, 1} for paragraph e, and {1, 4, 3, 2, 5} for paragraph c. This means that the sequence of paragraphs used to embed watermark will be

```
x = x_i;
for l = 1; l ≤ x_i; l = l + 1 do
    oldindex[l] = l;
end
j = θ_i^K (mod x!);
q = 1;
if x > 0 then
    s = ⌈ j / (x−1)! ⌉;
    j = j (mod (x − 1)!);
    newindex[q] = oldindex[s];
    for l = s; l ≤ x − 1; l = l + 1 do
        oldindex[l] = oldindex[l + 1];
    end
    q = q + 1;
    x = x − 1;
end
```

Algorithm 19: Sentence sequence generation

$$\left\{\{\rho^{x_1!}_{(\theta_1)^K \pmod{x_1!},1},\ldots,\rho^{x_1!}_{(\theta_1)^K \pmod{x_1!},x_1}\},\ldots \atop \{\rho^{x_y!}_{(\theta_y)^K \pmod{x_y!},1},\ldots,\rho^{x_y!}_{(\theta_y)^K \pmod{x_y!},x_y}\}\right\} \qquad \left\{\{t_{(1,1)},t_{(1,2)},\ldots,t_{(1,x_1)}\}, \atop \{t_{(y,1)},t_{(y,2)},\ldots,t_{(y,x_y)}\}\right\}$$

Table 4.2: Pseudo-randomization of watermarking sequence

Number of elements	Keys searched	
	Empirical result	Theoretical Value
2	1.88	2
3	4.44	4.5
4	7.33	10.6667
5	16.16	26.04171
6	64.27	64.8
7	145.22	163.401
8	516.55	416.102
9	1140.77	1067.63
10	3381.77	2755.73
11	6240.94	7147.66
12	15307.72	18613.9
13	37694.88	48638.8
14	108803.61	127463
15	433622.72	334865
16	1097114.16	881658
17	2004049	2330000
18	6203832.22	6150000
19	16376576.78	16300000

Table 4.3: Comparison of empirical results with theoretical values

paragraph d, then paragraph a, b, e and finally c. While using d (which contains 3 sentences) the sequence of sentences used for watermark embedding will be sentence 2, sentence 1 and finally sentence 3; and so on for sentences in other paragraphs.

For generating a permutation of a set containing y elements, the first element can be chosen in y ways, the second in $(y-1)$ ways and so on, and the total combinations (with repetitions) are y^y, hence, the probability of getting a permutation when choosing elements with repetitions is given be the following equation.

$$P(\theta_i \neq \theta_j, \forall i, \forall j, i \neq j) = (\frac{y!}{y^y}) \tag{4.2}$$

Results of the experiments conducted to generate permutations using AES-128 confirm the results. Table 4.3 compares empirical results with theoretical values.

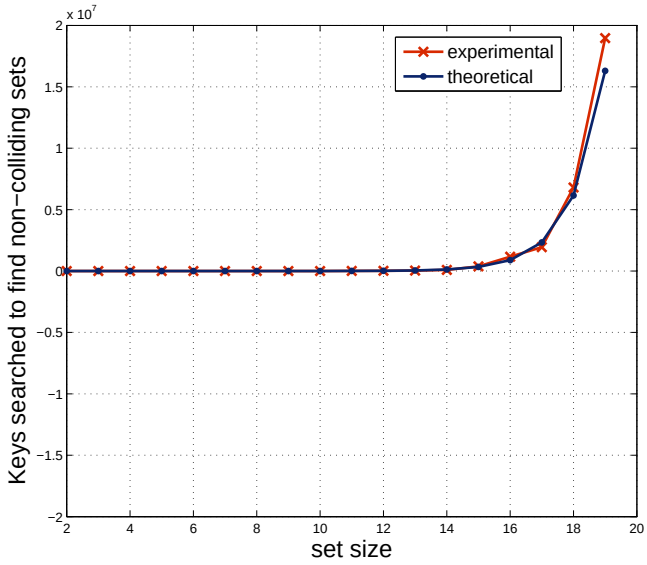

Figure 4.2: Keys required to get a valid permutation using AES-128

4.3.2 Watermark Composition

While constructing the watermarking, it is important to consider the following requirements.

1. Watermark can identify the publisher and user successfully.
2. Publisher cannot frame an innocent user.
3. Watermark can withstand collusion attacks.

To satisfy the first requirement, the watermark simply needs to have two components - a publisher component and a user component. But by this method, the publisher can generate any desired watermark and thus frame an innocent user. Hence we adopt the following protocol.

1. The publisher sends user a watermark W_u carrying the user identity.
2. User signs W_u with his private key Pr_u and sends publisher the signed user component $S_{Pr_u}(W_u)$.

3. Publisher verifies the correctness by verifying the signed user component with the user's public key Pu_u. He then appends the document specific publisher component W_p to the signed user component and signs it with his private key Pr_p.

4. Final watermark W_i is $S_{Pr_p}(W_p||S_{Pr_u}(W_u))$.

The court can verify the watermark with the public keys of publisher and user. Neither the publisher, nor the user can tamper with the watermark without the knowledge of the other person's private key. A small problem with this scheme is that since the user components of various users will differ, hence multiple users can collude and destroy the watermark. Hence W_u needs to be encoded such that colluding users can successfully be identified. For this purpose, we use the logarithmic length *c-secure* codes proposed in [16]. These codes can successfully identify at least one of the c colluding users from a group of n users. Given integers N and c, and an error tolerance metric $\epsilon > 0$, set $n = 2c$, $L = 2c\log(2N/\epsilon)$, and $D = 2n^2\log(4nL/\epsilon)$. The code $\Gamma'(L, N, n, d)$ (details in [16]) is $c - secure$ with ϵ-error.

Let the codeword for the user for which the document is being watermarked be $W_u = \{w_1, w_2, \ldots, w_{Ld(n-1)}\}$. The Boneh-code enables us to identify colluding parties of at most $c = n/2$ users with a probability of $1 - \epsilon$. For further details about these collusion-secure codes, please refer to [16]. Now the watermark $S_{Pr_p}(W_p||S_{Pr_u}(W_u))$ satisfies all three requirements mentioned at the beginning of the section and can be embedded.

4.3.3 Watermark Insertion

Before proceeding to the watermark insertion algorithm, we describe how watermark bits will be physically carried in the document. Let the number of words in a sentence s_i be d_i and the binary representation of d_i be $d_{i,1}, d_{i,2}, \ldots, d_{i,z}$ such that $d_{i,1}$ is the LSB. We utilize $d_{i,1}$ and $d_{1,2}$ to carry the watermark. If we want to embed two bits w_1 and w_2 in a sentence s_i, then,

1. Set $d_{i,1} = w_1$, $d_{i,2} = w_2$. Let the new value of d be d'.

2. Transform the sentence such that it contains d' number of words using one or more of the following (and other) transformations,

(a) Change of voice from active to passive and vice versa. Example, *"The cops arrested Teju"* ↔ *"Teju was arrested by the cops"*.

(b) Addition/deletion of an adjunct to/from the sentence. Example, *"The company praised Reema"* ↔ *"**It was the** company which praised Reema"*.

(c) Addition/Removal of optional articles. Example, *"Owen was cutting up the trees for Christmas"* ↔ *"Owen was cutting up trees for Christmas"*.

(d) Grouping of multiple subjects. Example, *"Meeta married Rinkle"* ↔ *"Meeta and Rinkle got married"*.

(e) Addition/removal of coordinate conjunctions. Example, *"Maya started to sing **and** Anjali began playing the guitar"* ↔ *"Maya started to sing, Anjali began playing the guitar"*.

(f) Introducing, or eliminating *"then"* from the **if ... then** pair of correlative conjunctions. Example *"If this is what Gautam wants, **then** this is what he'll get"* ↔ *"If this is what Gautam wants, this is what he'll get"*.

Given the information on how we are going to store watermarking bits in the document, the watermark embedding algorithm is given below.

1. All the sentences are marked as *unused*.

2. Choose the paragraphs corresponding to the next τ *unused* indices from the new paragraph sequence. (Go to start of sequence if end of sequence reached).

3. Take the first available *unused* sentences (using new sentence sequence) from the τ paragraphs and embed the first γ bits in them using Algorithm 20, where each watermark bit is inserted $\mu = \frac{\tau\beta}{\gamma}$ times. The watermark bit is physically embedded using English language transformations (discussed in [14]). For example, for a sentence *"This is not so difficult to understand"* having word count of 7 (0111) if we need to reduce one word from it to embed the watermark bits (10) in the 2 LSBs of its word count, preserving its meaning, we can change the sentence to *"Understanding this is not so difficult"* which has a word count of 6 (0110).

   It should be noted that the details of computational feasibility of performing English language transformations, including loss of quality in terms of meaning is out of scope of our research and has, therefore, been omitted. It is a topic

of interest for researchers in natural language processing (NLP) and it is our understanding that numerous research projects are addressing this issue. In our project, we transformed the sentence after manual inspection.

4. Delete the γ watermark bits embedded in the first step from the watermark.

5. Mark the sentences chosen in step 2 as *used* and if all the sentences of a paragraph are marked as *used*, mark the paragraph as *used*.

6. Repeat steps 2-5 till the entire watermark is embedded.

The pseudo-code for the above procedure is provided in Algorithm 20.

```
1  counter = 1;
2  for l=1 to 1 to y − 1 in steps of 1 do
3      q_l = {s_{t_(l,1)}, s_{t_(l,2)}, ..., s_{t_(l,x_l)}};
4  end
5  Q = {q_1, q_2, ..., q_y};
6  for i = 1; i ≤ m; i+ = β do
7      for j = 1 to τ in steps of 1 do
8          temp = (j + counter)( (mod y));
9          s_{t'_j} = s_{t_(temp,1)};
10         b_j = |s_{t'_j}|;
11         q_temp = q_temp − s_{t_(temp,1)};
12         if q_temp = φ then
13             Q = Q − q_temp;
14             y = y − 1;
15         end
16     end
17     for j=1;j ≤ τ/2;j++ do
18         for l=1;l ≤ β;l = l + 1 do
19             b_jl = w_{((j+l−1) (mod γ))+((counter−1)×γ)};
20             b_{(j+τ/2)(β−l+1)} = b_jl;
21         end
22         transform sentences according to new d by applying English
           language transformations;
23     end
24     counter = (counter + τ)( (mod y));
25 end
```

Algorithm 20: Natural language watermark insertion

Copy	Watermark bits				
	w_0	w_1	w_2	w_3	w_4
1	0	0	1	***1***	1
2	***1***	0	1	0	1
3	0	0	***0***	0	***0***
4	0	0	1	0	1
5	0	***1***	1	***1***	1
output	$w_0 = 0$	$w_1 = 0$	$w_2 = 1$	$w_3 = 0$	$w_4 = 1$

Table 4.4: Illustration of majority voting

4.3.4 Watermark Extraction and Verification

First, the permuted sequence of sentences is generated as in insertion (Algorithm 19). We then extract the watermark bits from the LSBs of the sentences identified as carrying the watermark bits. Finally, *majority-voting* is applied on the multiple instances of each watermark bit. Table 4.4 shows working of majority voting.

The extracted watermark W_e is compared to the inserted watermark W_i and if the Hamming Distance is less then a maximum tolerance value Ω, the watermark is acceptable, otherwise it is rejected (collusion detection is performed using algorithm suggested in [16]).

4.4 Analysis

4.4.1 Attacks

We discuss the various attacks possible on the watermarked document and degree of resilience offered by our scheme.

1. **Reformatting/Reproducing attacks**: The watermark is carried in the structure of the sentences and not the formatting information (such as interword or interline spacing, font characteristics, indentation, etc). Hence, changing these attributes does not alter the watermark.

2. **Sentence addition/ deletion**: Addition or deletion of a sentence s_i results in the sentence sequence being distorted for the paragraph (say p_j) containing s_i. But each watermark bit carried in sentences of p_j is embedding in $\mu - 1$ sentences in other paragraphs and can be correctly extracted using majority

voting (explained in 4.3.4). Hence, the watermark can withstand this attack. In the worst case, if the attacker adds or deletes $\frac{\mu}{2}$ sentences carrying the same watermark bits, the watermark might be destroyed. Thus, the watermark can survive at least $\frac{\mu}{2} - 1$ additions/deletions (**lower bound**).

3. **Text swapping**: Text swapping refers to selecting two sentences $s_i \in p_j$ and $s_{i'} \in p_{j'}$ from a document and swapping them. The sentence sequence is not disturbed in this case and only watermark bits corresponding to the swapped sentences are affected. Like in sentence addition/ deletion, the other $\mu - 1$ instances of the watermark bits result in correct watermark retrieval. Here also, the watermark can withstand at least $\frac{\mu}{2} - 1$ swaps.

4. **Paragraph shuffling**: In 4.3.1, we first sort the paragraph sequence according to cardinality before carrying out the permutation operation. Hence, even if the paragraphs are shuffled by the attacker, the original permutation will be restored when extracting the watermark. Hence, the scheme is totally secure against paragraphs being shuffled.

5. **Collusion attack**: Boneh-code is inserted as the user component W_u. If an illegal copy is discovered, then the algorithm described in [16] is executed which outputs the member(s) of the collusion with high probability.

6. **Cryptographic attacks**: AES lies at the core of our scheme as it is used to generate permutations. First the attacker needs $\mathcal{O}(2^k)$ time to perform an exhaustive search on K. For each potential K, however, the attacker would need to generate potential index sets, which requires $\mathcal{O}(2^k)$ time. Hence the time complexity of an exhaustive search attack is $\mathcal{O}(2^{2k})$. More importantly, a key K' different to key used to embed the watermark (K) can still, with high probability, generate a valid permutation. This permutation is different to permutation generated while watermark embedding and this introduces an uncertainty effect where the attacker cannot be sure of the correctness of a permutation generated by a random key.

4.4.2 False Positive Probability

The probability of an m-bit watermark matching another watermark extracted from a randomly picked document is 2^{-m}. Since each bit of the watermark is actually embedded at μ positions, $\frac{\mu}{2} + 1$ of those μ bits should match corresponding bit

of our watermark. This makes the actual probability of having *False Positives* = $2^{-(m+\frac{\mu}{2}+1)}$. This is lower than probability of false positives in [14].

4.4.3 Watermarking Capacity

The optimal capacity utilization is when a document contains $\sum_{j=1}^{y} x_j$ sentences and each sentence carries β bits. Every watermark bit is embedded in $\mu = \frac{\tau\beta}{\gamma}$ sentences. Hence the watermarking capacity of our scheme is $\frac{\beta \times \sum_{j=1}^{y} x_j}{\mu} = \frac{\gamma \times \sum_{j=1}^{y} x_j}{\tau}$.

4.5 Experimental Results

4.5.1 Implementation Details

The experiments were carried out in Unix using C language on Pentium-4, 2.4 GHz processor. Usage of C as a programming language makes the implementation extremely efficient in terms of time. Quartz digital signature scheme was utilized for producing digital signatures since the size of these signatures is very small (128-bits). Java implementation provided by Wolf [94] was used to generate signatures. Transformation of sentences to embed the watermark bit was done manually. The automation of the process of transforming sentences is an NLP topic and out of our research scope.

4.5.2 Results

We used 5 sample documents of varying sizes (from 16505 words to 46271 words) and paragraph structures to embed watermarks of 5 sizes constructed using quartz digital signature scheme (which produces 128-bit digital signatures) and analyzed the results of the experiments. It should be noted that the watermark embedded essentially consists of user's and publisher's signatures) and optionally other information like timestamp, metadata, padding and so on. The number of bits that change are proportional to the watermark size as indicated in Table 4.5.

The net change in document size is fairly constant for a specific document. The change in document size is less than 1% in most of the cases (refer to Table 4.6). Hence, quantitatively speaking, there is minimal distortion to the document. It was observed that the documents with larger paragraphs had fewer changes as compared to documents with smaller paragraphs. This also suggests that the paragraph

Watermark Size	Bit Changes				
(in bits)	document 1	document 2	document 3	document 4	document 5
320	1802	1762	1431	1269	1280
400	1903	1895	1507	1436	1334
480	2003	2037	1589	1522	1438
560	2182	2121	1657	1631	1526
640	2301	2266	1717	1726	1604

Table 4.5: Text modification with increasing watermark size

Watermark Size	Words Added				
(in bits)	document 1	document 2	document 3	document 4	document 5
320	-8	8	0	1	-14
400	-11	2	-4	-16	-12
480	-15	-5	-17	-1	-19
560	-14	-5	-20	-10	-26
640	-11	-7	-24	17	-14

Table 4.6: Text amplification with increasing watermark size

structure, and thereby the permutation we select play a key role in determining the number of words that will be added or deleted from the document.

4.6 Conclusion

Our scheme is shown to be resilient against document reproduction, reformatting, synonym substitution, text addition, text deletion, text swapping and paragraph shuffling. Previous watermarking schemes [18, 19, 24, 25, 54, 56, 65, 67, 98] are not secure against majority of these attacks. Compared to [14], our scheme provides higher security (deterministic resilience to at least $\frac{\mu}{2}-1$ changes against probabilistic resilience to single change in [14]) against text addition, text deletion, text swapping and total security against paragraph shuffling. It is also secure against collusion attacks through the adoption of Boneh-codes. An exhaustive cryptographic attack on the scheme takes $\mathcal{O}(2^{2k})$ time (k being the size of key used). With high probability, the scheme can successfully identify at least one of the colluding users in event of a collusion attack. The capacity of the scheme is $\frac{\gamma \times \sum_{j=1}^{y} x_j}{\tau}$ watermark bits.

Areas of improvement and future topics of research in this field include the following.

1. Increasing the capacity of the scheme by using an error correcting code instead of the currently used repetitive correcting code or majority-voting: In the existing scheme, each watermark bit is embedded in multiple paragraphs making it a repetitive code that reduces the watermark-carrying capacity of a document. Instead, if error-correcting codes are utilized, capacity would significantly improve.

2. Extending the scheme to multilingual documents incorporating the grammatical aspects of various languages: In the current implementation, only English documents are watermarked. Watermarking other documents would required analysis of grammar rules of that language. This is more of an implementation issue than a design issue as the underlying principle is the same.

Chapter 5

Software Watermarking

In 2005, Myles and Jin proposed a software watermarking scheme based on converting *jump instructions* or *unconditional branch statements* (UBSs) by calls to a *fingerprint branch function* (FBF) that computes the correct target address of the UBS as a function of the generated fingerprint and integrity check. If the program is tampered with, the fingerprint and integrity checks change and the target address will not be computed correctly. We propose an attack based on tracking stack pointer modifications to break the scheme and provide implementation details of the attack. The key element of the attack is to remove the fingerprint and integrity check generating code from the program after disassociating the target address from the fingerprint and integrity value. Using the debugging tools that give a control to the attacker to track stack pointer operations, we perform both subtractive and watermark replacement attacks. The major steps in the attack are automated resulting in a fast and low-cost attack.

Once the loopholes in previously proposed models were identified and the attack was successful, we were in a position to suggest a modified software watermarking model, which is resistant to such attacks. Detailed discussion on current software watermarking schemes was carried out in Section 3.3. We briefly revisit the current scenario here. Software watermarking is classified in the following categories.

- *Graph-based software watermarking*: The software is treated as a graph G_s with sequential blocks of code as nodes and transfer instructions such as function calls and branch statements as edges connecting the nodes. The watermark is a separate code and realized as a graph G_w. The two graphs G_s and G_w are connected by inserting additional edges (implemented as branch state-

ments). The resulting watermarked graph is $G_{s'} = G_s + G_w$ and source code s' is decoded from $G_{s'}$.

Venkatesan et al. [90] proposed the first graph-based software watermarking scheme. The central idea is to convert the software and the watermark code into digraphs and add new edges between the two graphs implemented by adding function calls between the software and watermark code. This scheme lacks error-correcting capabilities and is susceptible to re-ordering of instructions and addition of new function calls. Another problem in the scheme is that the random walk mentioned in their work (refers to the next node to be added in the watermarked software graph being selected randomly from the software graph and the watermark graph) is not actually *random*. The node visited next is based on the number of remaining nodes belonging to software graph N_s and the number of remaining nodes belonging to watermark graph N_w. The next node is chosen from the watermark nodes with probability of $\frac{N_w}{N_w+N_s}$ and from the software nodes with a probability of $\frac{N_s}{N_w+N_s}$. In a typical scenario, $N_s \gg N_w$, hence the watermark is skewed towards the tail of the watermarked program. This information is useful for probabilistic attacks. Alternatively, a pseudo-random permutation of the nodes to be visited can be generated. For further literature in graph-based software watermarking, the reader is referred to [26, 27, 28, 29, 88]. None of these schemes are secure against instruction and block re-ordering attacks.

- *Register-based software watermarking*: Registers used to store variables are changed depending on the watermark bit to be embedded by replacing higher level language code with an inline assembly code. The attacker intends to re-allocate variables in registers if the watermark has to be removed. Register-based software watermarking based on the QP algorithm (named after authors Qu and Potkonjak) [75, 76] is presented in [70]. It modifies registers used to store variables depending on which variables are required at the same time. The scheme is susceptible to register re-allocation attacks. A secondary watermark destroys the old watermark and inserting bogus methods renders the original watermark useless by changing the interference graph.

- *Thread-based software watermarking*: Nagra et al. [72] propose encoding the watermark in the sequence of the threads that are executed. For example, there are 3 threads; T_1, T_2, T_3, $T_1 \rightarrow T_2 \rightarrow T_3$ encodes watermark $(000)_2$ and

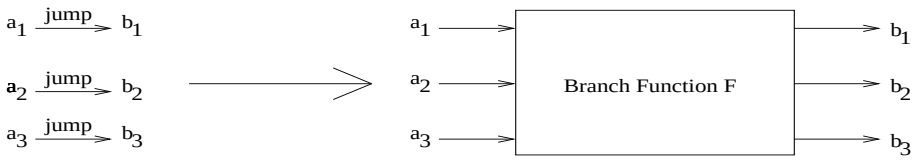

Figure 5.1: Branch function modifying return addresses

$T_1 \rightarrow T_3 \rightarrow T_2$ encodes watermark $(001)_2$ and so on. However, without any additional error-control mechanism, changing threads that execute piece of a code would destroy the watermark. Again, there has been no attack claiming to break the watermark using suggested approach.

- *Obfuscation-based software watermarking*: This class of watermarking is applicable to object-oriented software. Class C with functions $\{f_1, f_2, \ldots, f_n\}$ is partitioned into k subclasses $\{C_1, C_2, \ldots, C_k\}$ and the watermark is encoded in the allocation of the functionalities. Examples of such proposed schemes are given in [30, 39, 87].

- *Branch-based software watermarking*: Collberg et al. introduce the notion of *branch function* [26]. *Jump instructions* or *unconditional branch statements* (UBSs) are replaced by calls to the *branch function* (for the sake of consistency, by *branch*, we mean an unconditional branch statement from now on) and modifies its own return address in order to return the control to the target of the branch statement (Figure 5.1). If the program contains a branch from l_{begin} to l_{end}, several *pit stops* are added so that the control-flow graph becomes $l_{begin} \rightarrow a_1 \rightarrow a_2 \rightarrow \ldots \rightarrow l_{end}$ (l_{begin} has a jump instruction to a_1, a_1 has a jump instruction to a_2 and so on). Pit stops are inserted using following rule.

$$address(a_i) < address(a_{i+1}), \text{ if watermark bit } w_i = 1$$
$$address(a_i) > address(a_{i+1}), \text{ if watermark bit } w_i = 0$$

  Finally all the jump instructions are replaced by call to the branch function that determines the correct target address based on the calling address and returns the control to it.

  Obvious attacks on such a scheme are adding an additional pit stop or deleting an existing pit stop to disturb the chain (thereby modify the watermark) yet

keep the origin and target the same (hence keeping the execution path intact). Making similar changes, inserting secondary watermark is trivial.

Myles and Jin proposed an alternative fingerprinting model in [71]. The underlying concept remains the same, that is, a *branch function* transferring control to the target of the UBS, but in this case, the branch function contains the fingerprint-generating code, hence the name *Fingerprint Branch Function* (FBF). FBF also computes an integrity check on the source code to ensure that it is not modified. In the following section, we discuss this scheme in detail and analyze its flaws and weaknesses.

5.1 Description of Myles and Jun Watermarking Scheme

Branch statements are replaced by calls to an FBF which returns control to the target address. The target address is generated through a recursive process of deriving new key from previous key and checking the program for integrity. Additionally, an integrity check branch function (ICBF) is inserted in the program that verifies the integrity of FBF. When a program is manipulated, the keys derived and/or integrity check value change and hence the target address changes as well. The modified target address can be valid (belonging to code section of the program), which will result in incorrect execution of the program, or it can be invalid (lying outside the code section) resulting in a runtime error. Thus the program is secured against manipulation. We now discuss the two algorithms in the scheme, *embed* that inserts the watermark in the software and *recognize* that extracts the watermark from the watermarked software.

1. embed$(P, AM, key_{AM}, key_{FM}) \rightarrow P', FM$
2. recognize$(P', key_{AM}, key_{FM}) \rightarrow AM, FM$

where

- P is the original software,
- AM is the authorship mark,
- key_{AM} is the secret input sequence to generate a trace of the program used to embed the watermark - the same for all copies of watermarked software,

- key_{FM} is an initial secret key for deriving further keys; different for each copy of the watermarked program,
- FM is the fingerprint mark,
- P' is the watermarked software

5.1.1 Watermark Insertion

The steps involved in the ***embed*** algorithm are as follows.

1. Let α be the set of all functions in P. Run the program with a secret input sequence key_{AM}.
2. Obtain set F of functions that lie in the execution path when the program is run with input sequence key_{AM}, let $\beta = \alpha - F$.
3. The number of UBSs in functions that belong to F is n and the number of UBSs in functions from β is m.
4. Insert the two integer arrays; T of size n and R of size m in the data section of the program.
5. Compute displacement d_i between source s_i and target t_i of UBSs in functions that belong to F, so for instructions of the form $s_i : \quad jmp \quad t_i$, the displacement $d_i = t_i - s_i$
6. In the program P, insert FBF ξ that performs the following tasks,
   (a) Initializes $k_0 = key_{FM}$.
   (b) For $1 \leq i \leq n$,
      i. Computes integrity check value v_i.
      ii. Computes key k_i from (k_{i-1}, v_i, AM) by applying a one-way hash function SHA_1.

$$k_i = SHA_1[(k_{i-1} \bigoplus AM) \| v_i] \tag{5.1}$$

   (c) Stores d_i at $h(k_i)^{th}$ location in array T ($T[h(k_i)] = d_i$), where h is a hash function, $h : \{k_1, k_2, \ldots, k_n\} \rightarrow \{1, 2, \ldots, m\} (n \leq m)$.

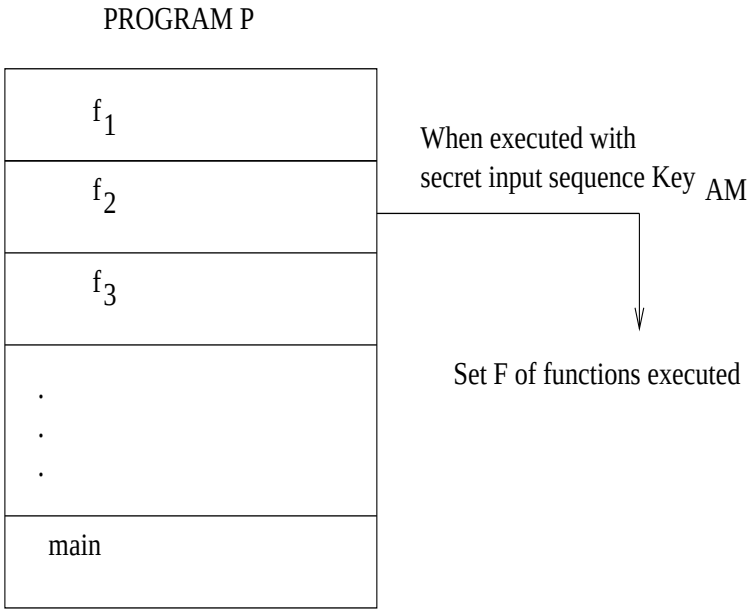

Figure 5.2: Function set F invoked using secret input parameter key_{AM}

7. Compute displacements e_i between source s_i and target t_i of UBSs in functions that belong to β.

8. Insert ICBF ϕ in the program that,

   (a) Computes integrity check value v_i. This value confirms the integrity of code section of the program containing ξ.

   (b) Stores displacement e_i in array R at index computed as a one-way hash function of v_i ($R[h(u_i)] = e_i$). The hash function h is the same that was used in Step 6.(c).

9. Replace all UBSs in F by calls to ξ and UBSs in β by calls to ϕ.

The fingerprint is generated as the embedding process executes. The final fingerprint is combination of all derived keys - $FM = k_1 \| k_2 \| \ldots k_n$. Users u_i, u_j have distinct initializing keys key_{FM_i}, key_{FM_j}, hence final fingerprints FM_i, FM_j are different.

5.1.2 Watermark Recognition

The *recognize* algorithm is run with the inputs P', key_{AM}, key_{FM} and outputs the authorship mark AM and fingerprint mark FM. When the program is run with the secret input key_{AM}, the function set F is executed, which generates the fingerprint mark $FM = k_1 \| k_2 \| \dots k_n$ by initializing $k_0 = key_{FM}$ and deriving successive keys using Equation 5.1. The authorship mark AM can be extracted by isolating the one-way hash function $k_i = SHA_1[(k_{i-1} \bigoplus AM) \| v_i]$.

5.2 Proposed Attack

Objective of the attacker is to convert the fingerprinted program P' to the original program P. Since the displacements in T are permuted, determining the correct target address of UBSs is computationally infeasible. Even if the size of T is small, the program can have error-guards that intentionally corrupt the program after a specific number of run-time errors, making hit-and-trial attack impossible. The function ϕ checks the integrity of ξ, adding to the security of the scheme and thereby making the attack more difficult.

In ξ, the integrity check is done and a key is generated. The key is then mapped to the index in the displacement array where the correct displacement is stored. Security of the scheme depends on the correct execution path being a function of keys and integrity checks. If the key generated or the integrity value is incorrect, the displacement is wrong, and therefore the execution path is wrong. We concentrate our attack on this dependence. As soon as we can *disassociate* the correct execution path from the keys and integrity check, the code generating keys and integrity check can be deleted. The authors of [71] claim that the attacker needs to analyze the data section of the program to notice any changes and read the displacement array. This claim is fallacious as an attacker can track register values, including the stack pointer (SP) at,

1. Entry point of ξ: SP = sp_{i_1}
2. Exit/ Return instruction of ξ: SP = sp_{i_2}

The difference $sp_{i_2} - sp_{i_1}$ gives the displacement value d_i. Identification of the instructions participating in fingerprint generation is also achievable. According

to [71], *"In the second phase of the algorithm, the branches in each function f that belongs to F are replaced by calls to the FBF"*. We can create a mapping of functions being called by other functions and thereby create sets of functions which all point to one particular function. ξ can be identified by the stack-pointer modifying statements and the set F can be identified as the set of functions calling ξ. Therefore, key_{AM} is no longer required to identify the set of functions participating in watermarking. Within the set F, each instruction calling ξ and having memory address sp_1 can now be replaced by an unconditional branch to the instruction at sp_2. This can be achieved using inline assembly programming. For example, in C++, a user can make use of inline assembler *_asm* [2]. As a result, the displacement and hence the correct target address is no longer a function of the key and integrity check. An example of such a block modifying the stack pointer is given below,

```
_asm {
        1:      pop ECX;
        2:      add ECX,dis;
        3:      push ECX;
}
```

In the above code, statement 1 extracts the current value of Stack Pointer into register ECX. Statement 2 adds the intended displacement *dis* to the popped value and statement 3 pushes back the modified value onto the Stack. The Stack Pointer now contains a modified return address. If *dis* is positive, the new address a_t is greater than the original return address a_r ($a_t > a_r$) and the control is transferred *forward*. If it is negative ($a_t < a_r$), control is transferred *backwards*. Observe that ϕ calls can similarly be replaced by the original UBSs.

After changing calls to ξ and ϕ by UBSs, the two functions (ξ, ϕ) can be deleted. When the *recognize* algorithm is run with input key_{AM}, key_{FM}, the inputs are unused dead variables, the algorithm doesn't output the fingerprint mark FM and the recognition algorithm fails. The resulting software is equivalent to an unwatermarked software.

Summarizing this process, the steps performed by the attacker are as follows.

1. *Identify* ξ: This task is accomplished by locating stack-pointer modifying state-

ments. For example, in C/C++, searching for *_asm* blocks. If a program contains multiple *_asm* blocks, the ones with modification operation on ESP (Stack Pointer) requires to be targeted.

2. *Identify* F: After identifying ξ, the fact that only the functions that belong to F call ξ can be utilized to identify F.

3. *Displacement computation*: Stack pointer values are recorded at the entry and exit points of ξ (sp_{i_1} and sp_{i_2} respectively) and displacement d_i is equal to $sp_{i_2} - sp_{i_1}$. Target instructions are determined from calling instruction and displacement. In our implementation, we use breakpoints to track the register values.

4. *Replacement of* ξ *calls to UBSs*: If the purpose of the attack is to remove the watermark, the function calls to ξ are replaced by UBSs to obtain the original watermarked code.

5. *Creating a modified watermarked program*: The attacker can embed his/her own authorship mark AM' after removing the original authorship mark AM. For a successful attack, (AM', FM') should be recognized on running *recognize* algorithm with parameters P', key_{FM}, key_{AM} where $FM' \neq FM$.

   (a) For all f that belong to F, compute the displacement between the calling address and the target address and store in an array along with the calling address.

   (b) Replace the UBSs by call to a new Fingerprint branch Function, $\widetilde{\xi}$.

   (c) $\widetilde{\xi}$ **need not** compute integrity check but simple calculates a new key based on the old key and attacker's authorship mark AM'.

   $$k_i = SHA1[k_{i-1} \bigoplus AM']. \tag{5.2}$$

   Comparing (1) and (2), $k'_i \neq k_i$, $1 \leq i \leq n$.

   (d) Map the keys to correct displacement using hash,
   $h : \{k'_1, k'_2, \ldots, k'_n\} \rightarrow \{1, 2, \ldots, m\} (n \leq m)$
   $T[h(k'_i)] = d_i$

The key sequence FM' generated is different from the original key sequence FM as the individual keys are different shown by the following proof.

$k'_1 \neq k_1, k'_2 \neq k_2, \ldots, k'_n \neq k_n$

$\Rightarrow \{\mathtt{k'_1}, \mathtt{k'_2}, \ldots, \mathtt{k'_n}\} \neq \{\mathtt{k_1}, \mathtt{k_2}, \ldots, \mathtt{k_n}\}$

$\Rightarrow \{\mathtt{k'_1}, \mathtt{k'_2}, \ldots, \mathtt{k'_n}\} \neq \mathtt{FM}$

$\Rightarrow \mathtt{FM'} \neq \mathtt{FM}$

The recognition algorithm now outputs FM', AM' when executed with the inputs P', key_{AM}, key_{FM}.

In terms of efficiency, the overall complexity of attack depends on complexities of steps 3 and 4 as others are one-off steps. Steps 3 and 4 have linear complexity and hence the attack has $\mathcal{O}(n)$ complexity. Steps 1 and 2 are automated and no human inspection is required to identify ξ and F.

5.3 Implementation Details and Results

We have implemented the watermarking scheme in Visual C++ and carried out the attack using the same. The features useful in doing so are the debug lookup windows - disassembly and register. The stack pointer value can then be tracked by using breakpoints under debugging mode and there is minimal manual intervention or inspection required. The following is disassembled code of the watermarked program used to compute displacement values.

Function f_i that belongs to F calling FBF ξ in statement 94:

```
0041198C  rep stos dword ptr es:[edi]

0041198E  mov eax,dword ptr [a]

00411991  cmp eax,dword ptr [b]
```

```
00411994  jle greater+2Bh (41199Bh)

00411996  call fingerprint (411271h)

0041199B  push offset string " is greater \n" (4177A8h)

004119A0 mov esi,esp

004119A2 mov eax,dword ptr [b]

004119A5  push eax 004119A6 mov ecx,dword ptr [__imp_std::cout
(41A350h)]

004119AC call dword ptr
          [__imp_std::basic_ostream<char,
          std::char_traits<char>>::operator<< (41A354h)]

004119B2  cmp         esi,esp

004119B4 call @ILT+425(__RTC_CheckEsp) (4111AEh)

004119B9  push eax 004119BA call
std::operator<<<std::char_traits<char> > (411168h)

004119BF  add esp,8 004119C2  jmp         l1+27h (4119EBh)

004119C4 push offset string " is greater \n" (4177A8h)

004119C9  mov esi,esp

004119CB mov         eax,dword ptr [a]
-----------------------------------------------------------------
Fingerprint branch\index{branch} function code modifying return address:

00414AF2  mov         eax,ebp
```

```
00414AF4  add         eax,4

00414AF7 mov         ebx,esp

00414AF9 mov         esp,eax

00414AFB pop ecx

00414AFC  sub eax,eax

00414AFE add         eax,0Ah

00414B01 add ecx,dword ptr [dis (419334h)]

00414B07  push ecx 00414B08  mov esp,ebx
-----------------------------------------------------------------
```

Register values are tracked while the program is executed and the following results are obtained,
Statement 00414AF2: EIP stores calling address, EIP=00411996.
Statement 00414AFB: Return address, stored in the stack pointer, is popped into ECX, ECX = 0041199B.
Statement 00414B01: ECX adds displacement value to calling address, ECX = 004119C4.
Statement 00414B07: ECX value is pushed onto stack pointer. *fingerprint();* returns control to this address.

```
-----------------------------------------------------------------
```

In a nutshell, instruction 94 calls *fingerprint();* which returns control to instruction 98 (the target of the original UBS) based on the value of *dis* looked up from array T. The attacker can thus compute the difference between ECX value at statement 80 (ECX_{80}) and ECX value at statement 83 (ECX_{83}) to find the value of displacement, then replace *fingerprint();* call at statement 94 by UBS transferring control to $\Psi(\Phi(94) + ECX_{83} - ECX_{80})$ (where $\Phi(x)$ denotes address of instruction x and

$\Psi(y)$ represents instruction at address y).

We have presented a successful low-cost attack on the branch-based watermarking scheme proposed in [71]. The cost of the attack is low in terms of hardware resources required since the only resources required are a functional computer with sufficient memory, storage and speed. The attack is efficient as manual inspection is required only during the step in which displacement values are noted from the disassembly register window. Even this is a debugger-specific constraint and in theory, it can be automated, however, we are unaware of an existing debugger that can perform this task. We provided an implementation of our scheme and some practical examples. The work lays a strong foundation for attacking similar software watermarking models [26, 27, 28, 29, 88] that depend on branching and inserting bogus functions in the program in order to embed a watermark. We also demonstrate that tracking registers and branches is a trivial task using debugging tools and hence opens up a very interesting question of how can the watermarking schemes survive attacks with such advanced capabilities? The next step, of designing a more secure software watermarking model deals with creating more complex dependency of inherent functionality of the program on the keys generated so that the attacker cannot remove fingerprint code without affecting the correct execution of the program. This can be done by introducing parameters other than displacement to bind the program's execution to the keys generated.

5.4 Surviving the Debugging Attack

The basic assumption we take is that the source code is available to the attacker for inspection. This is a strong assumption taking into account that most commercial softwares do not come with the source codes. However, we take into consideration the growing popularity of open source software as well. Plus, having a stronger assumption and thereby an easier attack, our watermarking scheme results in getting stronger (if it can survive the attacks). Manual inspection of the source code is practically infeasible, given that it can run into hundreds of thousands of code lines. Thus the attacker tries to minimize the size of source code that (s)he manually inspects. Debugging mechanisms provide strength to the attacks in such cases by reducing the size of code to be inspected to potentially a few hundred lines. According to [45], in order to identify $\mathcal{FBF}$, the attacker relies on either,

- $\mathcal{FBF}$ containing assembly level code or,
- Stack Pointer value differing at entry and exit of $\mathcal{FBF}$.

We have a typical scenario, where a source instruction I_s needs to transfer control to a target instruction I_t. I_s calls $\mathcal{FBF}$ which manipulates the stack pointer and returns the control to I_t. Addressing the first indicator, the code that performs stack pointer modifications can always be written in a higher language and thus it is not necessary that $\mathcal{FBF}$ contains assembly level code.

The stack pointer modification in $\mathcal{FBF}$ is the basis of attack. If $\mathcal{FBF}$ does not perform this extremely visible and conspicuous stack pointer modification and only returns the value of generated key to I_s, then I_s can add compute the displacement, add it to the stack pointer and transfer control to I_t. Given this process, the attack $\mathcal{FBF}$ cannot be identified and all subsequent steps of the attack fail. Another advantage is that the attacker can no longer get the values of all displacement values by placing two breakpoints at the start and end of $\mathcal{FBF}$ (which is the case in [71]). This can be achieved by shifting the stack pointer modification instruction from $\mathcal{FBF}$ to function containing I_s.

To get the values of displacements in the new model, the attacker would need to place breakpoints before and after each source instruction I_s. In the previous model, the attacker would have had to place only two breakpoints; at the start and end of $\mathcal{FBF}$ irrespective of the number of unconditional branch statements, and thus the amount of work attacker had to do was independent of code size. But now the amount of work attacker needs to perform manually is directly proportional to the number of unconditional branch statements present in functions from F.

Another strong assumption in our attack is that only unconditional branch statements in functions belonging to F call $\mathcal{FBF}$. If certain bogus calls to $\mathcal{FBF}$ are inserted while embedding the watermark, this assumption in not true. Thus identifying F is not possible for the attacker through methods pointed out by us. The action taken by $\mathcal{FBF}$ when called by these bogus statements is pre-defined during embedding.
The modified algorithm $embed_2$ is given below.

1. Let F be the set of functions that lie in the execution path when the program

is run with input sequence key_{AM}, let $\bar{F}$ be the set of the remaining functions in P.

2. Let the number of unconditional branch statements in functions in F and $\bar{F}$ be n and m respectively.

3. Add two arrays to the data section of the program - DT_F of size n and $DT_{\bar{F}}$ of size m.

4. Let the displacement between the source and target of the i^{th} unconditional branch statements in functions from F be $d_i = t_i - s_i$, where s_i is the source/origin and t_i is the target/destination.

5. In P, insert fingerprint branch function $\mathcal{FBF}$ which implements the following steps.

   (a) $k_0 = key_{FP}$.

   (b) For $1 \leq i \leq n$,

      i. Check integrity value $\mathcal{IC}_i$ of section $\mathcal{S}_i$ of code.

      ii. Generate k_i using $k_{i-1}, \mathcal{IC}_i, AM$ from the one-way hash function SHA_1.

$$k_i = SHA_1[(k_{i-1} \bigoplus AM) \| \mathcal{IC}_i] \tag{5.3}$$

6. Return k_i to the calling instruction I_s.

7. I_s looks up the displacement that is indexed by hash of the key and stored in table DT_F in data section of the program. $DT_F[h_1(k_i)] = d_i$, where h_1 is a hash function mapping the keys to the indices,

   $h_1 : \{k_1, k_2, \ldots, k_n\} \rightarrow \{1, 2, \ldots, \tilde{n}\} (n \leq \tilde{n})$.

8. I_s transfers control to $sp + d_i$ where sp is the current value of stack pointer, using higher language code.

9. Insert integrity check branch function $\mathcal{ICBF}$ in P that checks integrity $\mathcal{IC}'_i$ of code containing $\mathcal{FBF}$

10. Return $\mathcal{IC}'_i$ to the calling instruction $\bar{I}_s$.

11. $\bar{I_s}$ looks up the displacement that is indexed by hash of the key and stored in table $DT_{\bar{F}}$ in data section of the program. $DT_{\bar{F}}[h_2(\mathcal{IC}'_i)] = \bar{d_i}$, where h_2 is a hash function mapping the integrity checks to the indices,
$h_2 : \{\mathcal{IC}'_1, \mathcal{IC}'_2, \ldots, \mathcal{IC}'_m\} \rightarrow \{1, 2, \ldots, \tilde{m}\} (m \leq \tilde{m})$.

12. $\bar{I_s}$ transfers control to $sp + d_i$ where sp is the current value of stack pointer, using higher language code.

13. Replace unconditional branch statements in F and $\bar{F}$ by calls to $\mathcal{FBF}$ and $\mathcal{ICBF}$ respectively.

Steps 6–8 and 10–12 are modified such that the control transfer is shifted from $\mathcal{FBF}$ and $\mathcal{ICBF}$ to F and $\bar{F}$, respectively.

5.5 Analysis

Following are two major modifications to the watermarking algorithm of Myles and Jun.

1. strict usage of higher language code to perform stack pointer modifications.
2. shifting the stack pointer modification from the fingerprint branch function $\mathcal{FBF}$ to the source instruction of the unconditional branch statement.

Figures 5.3 and 5.4 illustrate the differences between the watermarking scheme of Myles and Jun and our proposed scheme. While in the former, $\mathcal{FBF}$ performs the key, displacement and target address computation, in the latter scheme it only performs key computation and returns the value of the key to the source instruction. The source instruction then computes target address as a function of displacement, which in turn is computed from the key. Thus an attacker has to place n pairs of breakpoints to find the correct target addresses. Thus, *manual* component complexity increases from $\mathcal{O}(1)$ in the previous scheme to $\mathcal{O}(n)$ in our proposed scheme. Security against other attacks such as additive or subtractive attacks remains the same as in [71].

5.6 Conclusion

We have presented modifications on the watermarking scheme from [71] so that the watermarking scheme can withstand debugging attacks like the one suggested

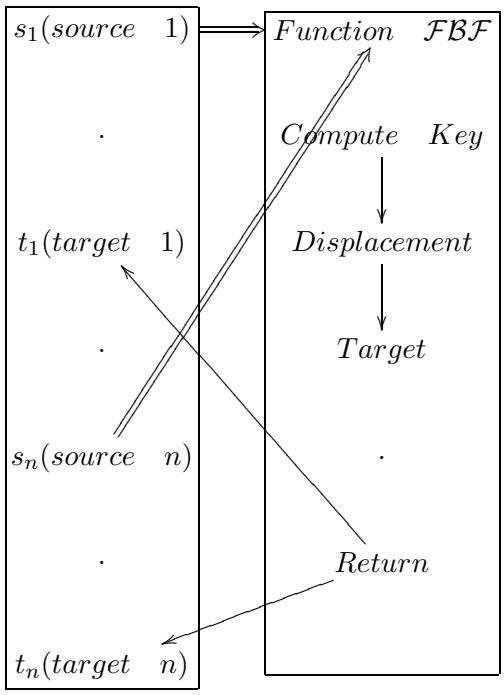

Figure 5.3: Fingerprint branch function modifies the return address itself

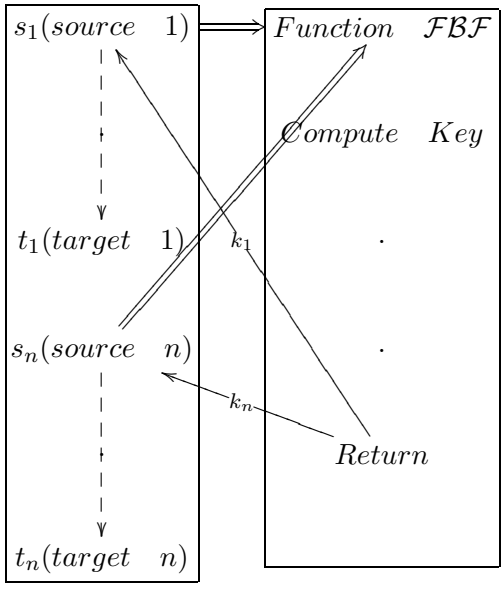

Figure 5.4: Calling instruction modifies address using key returned by fingerprint branch function

in [45]. The attacker needs much more extensive manual inspection ($\mathcal{O}(n)$) of the watermarked program in order to remove the watermark. This can prove to be infeasible given that software sizes can easily run into thousands of lines of code. The key to surviving the attack is shifting the stack pointer manipulation operation from the fingerprint branch function to the original unconditional branch statements.

The proposed watermarking scheme makes it more difficult and more tedious for an attacker to locate manipulative functions such as $\mathcal{FBF}, \mathcal{ICBF}$ but does not rule out eventual location and deletion of these functions. It is desirable to formulate a watermarking scheme belonging to the family of stack modifying functions that is completely secure against debugging attacks. To accomplish this goal, one needs to hide the dependency of target instruction calculation on key generation process from the user. This is an open problem in the field of branch based software watermarking.

Chapter 6

Semi-blind and Reversible Database Watermarking

6.1 Introduction

We discussed Agrawal and Kiernan's watermarking model in Section 3.4 and identified several shortcomings with their scheme which may lead to successful attacks on watermarked relations. One major concern was that the scheme lacked reversibility, that is, ability to revert back to the original relation from the watermarked relation. This leads to the possibility of successful secondary watermarking attacks. In this section we present a modified scheme that is reversible and semi-blind. We call the scheme semi-blind because it does not require the original database to detect watermark but the insertion algorithm stores the original bits selected for modification as an embed trace $\mathcal{ET}$, which is input to the detection algorithm. Size of $\mathcal{ET}$ is proportional to the number of tuples being marked. Previous watermarking schemes such as those studied in [86] have also presented similar semi-blind watermarking models. Our scheme is an enhancement of the irreversible watermarking model proposed by Agrawal and Kiernan [11]. We show that secondary watermark attacks are feasible on the schemes from [11]. Further, we modify the model to eliminate this shortcoming and propose an additional algorithm to identify the rightful owner from n contenders.

6.2 Related Work and Agrawal-Kiernan Scheme

Several relational database watermarking models have been proposed in [11, 12, 86, 40, 42, 61, 62, 101, 102, 103]. These schemes are irreversible with the exception of [102] and do not preserve queries (only the scheme described in [40] preserves query results). *Irreversible* watermarking implies that the original relation cannot be restored from the watermarked relation. Ownership disputes might be unresolved if an attacker successfully embeds a secondary watermark. But if the watermarking is reversible, the original database can be restored and the correct owner identified using suitable algorithm (discussed in section 6.4.1).

Agrawal and Kiernan [11] were the first to present a database watermarking scheme that modifies LSBs of numerical attributes (selected using the private key and tuple's primary key value). This *key-based attribute selection* is common to other proposals [12, 61].

6.2.1 Agrawal-Kiernan Scheme

In Section 3.4. we have discussed the Agrawal and Kiernan database watermarking scheme in details. In this section, we briefly re-visit the algorithm once again. The watermarking scheme consists of two algorithms; ***insertion***, and ***detection***. The bits modified during insertion are checked for correctness in the detection algorithm for establishment of watermark presence. Parameters to the insertion algorithm are as follows.

- Database relation R
- Number of tuples η
- Number of modifiable attributes υ, $\{A_0, A_1, \ldots, A_\upsilon\}$
- Number of modifiable LSBs ξ
- Fraction of tuples to be watermarked $1/\gamma$
- Private key $\mathcal{K}$

Let the secret parameter set given be $\phi = (\mathcal{K}, \gamma, \upsilon, \xi)$. Algorithm 21 illustrates the watermark insertion process. Tuples are selected using message authentication code (MAC) $\mathcal{F}(r.P)$ defined as $\mathcal{H}(\mathcal{K}\|\mathcal{H}(\mathcal{K}\|(r.P))$ [81] and appropriate bit

in the tuple set to $\mathcal{H}(\mathcal{K}\|r.P) \pmod 2$ till $\omega = \frac{\eta}{\gamma}$ bits are marked. Converse procedure is applied on the watermarked copy to detect the watermark (Algorithm 22) by verifying the modified bit is equal to $\mathcal{H}(\mathcal{K}\|\tilde{r}_w.P) \pmod 2$. The primary key value is unchanged. Parameters to the watermark detection algorithm are watermarked database relation R_w containing η tuples and υ attributes $\{A_0, A_1, \ldots, A_\upsilon\}$, number of LSBs modified ξ, fraction $1/\gamma$ of tuples watermarked, upper bound α on probability of falsely detecting watermark, minimum number τ of correctly marked attributes for successful detection, and private key $\mathcal{K}$.

Input: relation R, private key K, fraction $\frac{1}{\gamma}$, LSB usage ξ
Output: Watermarked relation R_w

```
1 forall tuple r ∈ R do
2     if F(r.P) (mod γ) = 0 then
3         i = F(r.P) (mod υ);
4         j = F(r.P) (mod ξ);
5         r.A_i^j = H(K‖r.P) (mod 2);
6     end
7 end
8 return R;
```

Algorithm 21: Watermark insertion [11]

Equation 6.1 gives the binomial probability of having at least k successes from n trials where probability of success in a single trial is p. During detection, at least τ bits need to be detected correctly in order to extract the correct watermark or in other words the probability of τ out of ω bits matching by sheer chance $B(\tau, \omega, \frac{1}{2})$ should be less than the upper bound α of false positive probability.

$$\mathcal{B}(k, n, p) = \sum_{i=k}^{n} \binom{n}{i} p^i (1-p)^{n-i} \tag{6.1}$$

6.2.2 Security Provided by Agrawal-Kiernan Scheme

While discussing the security of the scheme, Agrawal and Kiernan consider the following collection of attacks.

A1: *Bit flipping attack*: Updating some bits in numerical attributes.

A2: *Randomization attack*: Assigning random values to some bits.

Input: Watermarked relation $\tilde{R}_w$, private key K, fraction $\frac{1}{\gamma}$, LSB usage ξ
Output: detection Status $\in \{true, false\}$

$totalcount = matchcount = 0$;
forall *tuple* $\tilde{r}_w \in \tilde{R}_w$ **do**
  if $\mathcal{F}(r.P) \pmod{\gamma} = 0$ **then**
    $i = \mathcal{F}(r.P) \pmod{\upsilon}$;
    $j = \mathcal{F}(r.P) \pmod{\xi}$;
    if $\tilde{r}_w.A_i^j = \mathcal{H}(\mathcal{K} \| \tilde{r}_w.P) \pmod 2$ **then**
      $matchcount = matchcount + 1$;
    end
    $totalcount = totalcount + 1$;
  end
end
$\tau = min\{\theta : \mathcal{B}(\theta, totalcount, 1/2) < \alpha\}$; // $\mathcal{B}$ defined in Equation 6.1
if $matchcount \geq \tau$ **then**
  return $true$;
end
return $false$;

Algorithm 22: Watermark detection [11]

A3: *Rounding attack*: Rounding off a fixed number of bits.

A4: *Translation attack*: Transforming numerical values to another data type.

A5: *Subset attack*: Removing a small subset of tuples or attributes.

A6: *Mix and match attack*: Applying A4 on multiple relations and merging them.

A7: *Additive attack*: Re-watermarking an already watermarked relation.

A8: *Invertibility attack*: Checking if detection returns true for a random key.

Inserting new tuples to destroy watermark will not succeed as $\mathcal{F}(r.P)$ identifies marked tuple and two tuples cannot have the same primary key. Success of removing the watermark by deleting tuples depends on the parameter γ. Probability of destroying watermark by deleting a few tuples is extremely low when the fraction of tuples marked when γ is high. If γ is high for a fixed n, $1/\gamma$ is low and hence the fraction of tuples marked are low. Thus the probability of the attacker modifying the watermarked tuples is low. Bit flipping attacks (A1–A3) are probabilistically ineffective since the identification of correct tuples, attributes and LSBs is dependent on MAC. Additive and invertibility attacks are still feasible.

6.3 Analysis of Agrawal-Kiernan Watermarking Scheme

Based on our observations, Agrawal and Kiernan scheme has the following major weaknesses.

1. **Susceptibility of secondary watermarking:** Secondary watermarking refers to an attacker who is trying to insert his watermark in an already watermarked relation. The scheme does not protect against secondary watermarking as the attacker can choose his/her own parameter list $\tilde{\phi}$ and insert a new watermark in the original watermarked relation. The new watermark will establish the ownership of the attacker over the relation and might also destroy the original watermark. If the watermarking is reversible, the actual owner's watermark can be recovered from the reversed relation.

2. **Lack of query-preservation:** If an attribute $r.A_i = x_1$ is modified to x_2, then query **"Select** r **from** R **where** $r.A_i = x_1$**"** cannot be preserved. Thus, it is obvious that not all queries are preservable in watermarked database.

Currency code	**Nation**	**Buying rate**	**Selling rate**
AUD	Australia	133	125
INR	India	4500	4300
THB	Thailand	3740	3510
SLR	Sri Lanka	4430	4210
NZD	New Zealand	151	134

Table 6.1: Original foreign exchange rates relation

Currency code	**Nation**	**Buying rate**	**Selling rate**
AUD	Australia	133	125
INR	India	4500	4300
THB	Thailand	3740	3510
SLR	Sri Lanka	***4530***	***4310***
NZD	New Zealand	151	***124***

Table 6.2: Watermarked foreign exchange rates relation

Distance $\delta_{r.A_i}$, that refers to the minimum difference between value of $r.A_i$ from values of A_i in other tuples, is not considered in [11], due to which queries might not be preserved. If we change value of an attribute beyond it's distance, the ordering of the tuples is modified when the relation is sorted on that attribute and hence query results change. Consider the following relations that contains foreign exchange rate data of some countries against 100 US Dollars. Table 6.1 is the original relation and Table 6.2 is the watermarked relation. Result of queries **"Select *Nation* from *ForEx* where *Selling rate<130*"** and **"Select *Currency* from *ForEx* where *Buying rate is maximum*"** are different when executed on the original and watermarked relations.

3. **Lack of tolerance of attributes:** The number of LSBs that can be used for watermarking are not dependent on the *tolerance* of the attributes. This results in the possibility that the relation becomes unusable from a user's perspective. *Tolerance* is different from *distance*. For example, even if population of the two countries differ by millions, modifying population values beyond a

couple of thousands might render the data useless. Hence, the number of bits that one can change does not depend only on distance, but also on tolerance.

We propose the following modifications to eliminate each of these weaknesses.

1. **secondary watermarking** To defeat secondary watermarking attacks, the step $r.A_i^j = \mathcal{H}(\mathcal{K}\|r.P) \pmod 2$ in Algorithm 21 is changed to the following,

$$\mathcal{ET} = \mathcal{ET}\|r.A_i^j,$$
$$r.A_i^j = \mathcal{H}(\mathcal{K}\|r.P\|r.A_i^j) \pmod 2$$

   Bit $r.A_i^j$ is concatenated to embed trace $\mathcal{ET}$ and then modified. The scheme is semi-blind and reversible, since the original values can be restored from $\mathcal{ET}$. The size of $\mathcal{ET}$ is proportional to $\frac{n}{\gamma}$. At the detection time, the value of *matchcount* is incremented only if $\tilde{r}_w.A_i^j == \mathcal{H}(\mathcal{K}\|\tilde{r}_w.P\|\mathcal{ET}[totalcount]) \pmod 2$, where $i, j, totalcount, matchcount$ are counters updated during the detection.

   The owner stores $\mathcal{ET}$ at a secondary location. ($\mathcal{ET}$,$\mathcal{K}$) is the watermark detection key. Subsection 6.4.1 discusses how the rightful owner is identified if multiple parties watermark a relation in some sequence. Implementation is given in Algorithm 25.

2. **Query preservation** The value of an attribute $r.A_i$ should be modified by less than the distance $\delta_{r.A_i}$. Thereby, the number of bits available for watermarking are $\lfloor \log_2(\delta_{r.A_i}) \rfloor$ (For example, if the smallest difference between values of an attribute in two rows is 57.68, then only 5 bits can be used for watermarking as $\log_2(57.68) = 5.85$ and $\lfloor 5.85 \rfloor = 5$). This would guarantee query-preservation for the existing relation. Since the watermarking scheme is reversible, it facilitates incremental watermarking. The steps involved in incremental watermarking are,

   (a) Restore relation to unmarked version.

   (b) Add (or delete) required tuples (or attributes).

   (c) Re-watermark the updated relation.

3. **Tolerance** Since each attribute has a different tolerance limit beyond which it should not be modified, it is recommended that the number of LSBs to

utilize for watermarking should be a function of tolerance of the attributes. Hence, ξ_i LSBs of attribute A_i can be modified. The list of all these values $\Xi = \{\xi_1, \xi_2, \ldots, \xi_\upsilon\}$, where υ attributes are available for watermarking.

6.4 Modified Algorithms

With the above modifications, the secret parameter list for watermark detection becomes $\phi = (\mathcal{K}, \mathcal{ET}, \gamma, \upsilon, \Xi)$. We present a reversible and semi-blind watermarking scheme that comprises of the following three algorithms.

- *insertion*
- *detection*
- *owner identification*

The algorithms are presented in Algorithm 23, Algorithm 24, and Algorithm 25 respectively. They contain comments illustrating the purpose served by various steps. The acronym WM refers to *watermark* in the three algorithms.

6.4.1 Identifying Rightful Owner

In this additional algorithm, ownership disputes can be resolved through backtracking. If $R \xrightarrow{ins(p_1)} R_1$ is followed by $R_1 \xrightarrow{ins(p_2)} R_2$, then $R_2 \xrightarrow{det(p_2)} R_1$ will show that the restored relation R_1 has already been watermarked by another party (p_1) and hence p_2 is not the original owner. For all potential owners u_i, we compare relations restored $R_{restored}$ after detecting watermark of party u_i, and if it matches any other party's watermarked relation R_w within a preset tolerance limit ϵ, then u_i is eliminated from the list of possible owners. Each party supplies its secret parameter list ϕ. and the relation R_w on which it claims ownership.

We do not need all the watermarking parties to be traced and forced to particpate in the owner identification process. The algorithm can identify the rightful owner from a subset of the watermarking parties. Therefore, if the rightful owner O_R of a relation R gets the information about some parties who claim ownership on a similar version of R, then O_R can prove its ownership on R even without the knowledge of all parties who have watermarked different versions of R.

Input: relation R, private key K, fraction γ, number of markable attributes υ, LSB usage $\Xi = \{\xi_1, \xi_2, \ldots, \xi_\upsilon\}$
Output: Watermarked relation R_w, Embed Trace $\mathcal{ET}$

```
1  count = 0 ;                                   // index in WM to be generated
2  forall tuples r ∈ R do
3      if F(r.P) (mod γ) = 0 then
4          i = F(r.P) (mod υ);                   // identify attribute
5          j = F(r.P) (mod ξ)_i;                 // identify bit
6          if j < ⌊log2(δ_{r.A_i})⌋ then
7              ET[count] = r.A_i^j;              // store old value in WM
8              count = count + 1;                // next watermark bit's index
9              r.A_i^j = H(K‖r.P‖r.A_i^j) (mod 2) ;   // modify bit in
               relation
10         end
11     end
12 end
```

Algorithm 23: Reversible and semi-blind watermark insertion

6.5 Analysis

The attacker Mallory needs to flip at least $\bar{\tau} = \omega - \tau + 1$ marked bits to carry out a successful attack, where $\omega = \frac{\eta}{\gamma}$ [11]. Let us assume that Mallory somehow knows the values of ξ and υ and randomly chooses ζ tuples. The probability that this attack will succeed when Mallory flips A_i^ξ for all υ attributes in all randomly selected ζ tuples is given in Equation 7.4 [11], and the values are provided in Table 6.3. For our modified watermarked scheme, $\xi = \frac{\sum_{i=1}^{\upsilon} \xi_i}{\upsilon}$. Note that if the attacker flips more than 50% bits, the watermark will be detected when the all bits in the relation are flipped. This also gives us a fair idea about the value of γ that should be chosen. It should be fairly low and somewhere in between 10 and 100 as the attack is ineffective for values in this range.

$$\mathcal{P}(\mathcal{A}) = \sum_{i=\bar{\tau}}^{\omega} \frac{\binom{\omega}{i}\binom{\eta-\omega}{\zeta-i}}{\binom{\eta}{\zeta}} \tag{6.2}$$

Without the knowledge of ξ, γ, υ, Mallory's task is much tougher. To compensate for the lack of knowledge, Mallory might need to choose an estimated ξ'

Input: Watermarked relation $\tilde{R}_w$, Secret parameter list $\phi = (\mathcal{K}, \mathcal{ET}, \gamma, \upsilon, \Xi)$
Output: {Watermark Status $\in \{true, false\}$, Restored relation R}

```
1  R = R̃_w;
2  matchcount = 0 ;                          // matching WM bits counter
3  totalcount = 0 ;                          // total WM bits counter
4  forall tuples r̃_w ∈ R̃_w do
5      if F(r̃_w.P) (mod γ) = 0 then
6          i = F(r̃_w.P) (mod υ);             // identify marked attribute
7          j = F(r̃_w.P) (mod ξ)_i;           // identify marked bit
8          if j < ⌊log2(δ_{r̃_w.A_i})⌋ then
9              if H(K‖r̃_w.P) ⊕ ET[totalcount] (mod 2) = r̃_w.A_i^j then
10                 matchcount = matchcount + 1;   // bit authenticated
11                 r̃_w.A_i^j = ET[totalcount];    // restore bit in relation
12             end
13             totalcount = totalcount + 1;
14         end
15     end
16 end
17 τ = min(θ) : B(θ, totalcount, 1/2) < α;   // threshold check
18 if matchcount ≥ τ then
19     return {true, R};
20 else
21     return {false, R̃_w};
22 end
```

Algorithm 24: Reversible and semi-blind watermark detection

Input: Potential owners $\mathcal{U} = \{u_1, u_2, \ldots, u_n\}$. Secret parameter list of each u_i, $\mathcal{I}_{u_i} = \{\mathcal{K}_i, \mathcal{ET}_i, \gamma_i, \upsilon_i, \Xi_i\}$, tolerance ϵ, Potential owners' versions of the watermarked relation $\{R_1, R_2, \ldots, R_n\}$
Output: Owner O

```
forall u_i ∈ U do
    if detect(R_i, I_{u_i}) == {false, R'_i} then
        U = U \ u_i;
    end
    if detect(R_i, I_{u_i}) == {true, R_i^{rev}} then
        if {u_j : detect(R_i^{rev}, u_i) == {true, R_{temp}}, ∀j ≠ i} ≠ null then
            U = U \ u_i;
        end
    end
    return U;
end
```

Algorithm 25: Semi-blind owner identification

γ	**bits flipped**	**success probability**
10000	40%	0.64
1000	46%	0.44
100	48%	0.11
10	>50%	≈ 0

Table 6.3: Probability of success for bit flipping attack

and flip that bit of each of the attribute which degenerates the data quality. The security analysis for [11] also holds for our scheme as the underlying operations are retained.
The advantages of our reversible watermarking scheme as compared to [11] are as follows.

1. **Ownership resolution amongst n parties**

   A publisher releases its watermarked relation in the public domain, where n users distort the relation and embed their watermarks in the obtained relation. Each of these users then distribute their watermarked copies that other users then watermark and this process repeats. Thus we get a tree-like structure of watermarked relations. At some stage in future, the original publisher finds similar relations to his floating around in the public domain and takes the *owners* of these relations to court over copyright violation.

   If we model parties watermarking relations as nodes of a tree where the actual owner is the root of the tree, then the probabilities with which the owner will be correctly identified despite nodes from n levels of the tree abstaining from participation is given by $(1-\mathcal{P}(\mathcal{A}))^n$. These probabilities are calculated taking into consideration the modifications the attacker might make in the relation before watermarking it. The probability that an attacker will succeed in destroying the watermark is $\mathcal{P}(\mathcal{A})$ and hence the probability of the relation surviving an attack is $1-\mathcal{P}(\mathcal{A})$. The probability of a relation surviving n sequential attacks is $(1-\mathcal{P}(\mathcal{A}))^n$. It is extremely rare that the relation will be distributed beyond three or four levels as there usually a few companies dealing with similar data and furthermore distorting the data too much or too many times by a single entity would destroy the quality and usability of the data. It is shown in [11] that the attacker has a probability of 11% success if he changes 48% of the tuples assuming $\gamma = 100$. Hence, if only C and d_3 participate in the correct algorithm, C will be identified as the correct owner with a probability of 89% since parties from only one level (d_1, d_2) abstain. This probability is 100% if C, d_1 participate or C, d_2 participate. In general, successful detection of the correct watermark occurs with the probability of 0.89^n where n levels abstain from participation for $\gamma = 100$. Thus the probability of finding rightful owner if two levels abstain is $0.89 * 0.89 = 0.79$.

   Consider a company C that drags five data servers d_1, d_2, d_3, d_4 and d_5 to

	R_0	R_0^{rev}	R_1	R_1^{rev}	R_2	R_2^{rev}	R_3	R_3^{rev}	R_4	R_4^{rev}	R_5	R_5^{rev}
C	1	n.a.	n.a.	0.89	n.a.	0.89	n.a.	0.79	n.a.	0.79	n.a.	0.79
d_1	n.a.	σ	1	n.a.	n.a.	σ	n.a.	0.89	n.a.	0.89	n.a.	0.89
d_2	n.a.	σ	n.a.	σ	1	n.a.	n.a.	σ	n.a.	σ	n.a.	σ
d_3	n.a.	σ	n.a.	σ	n.a.	σ	1	n.a.	n.a.	σ	n.a.	σ
d_4	n.a.	σ	n.a.	σ	n.a.	σ	n.a.	σ	1	n.a.	n.a.	σ
d_5	n.a.	σ	n.a.	σ	n.a.	σ	n.a.	σ	n.a.	σ	1	n.a.

Table 6.4: Detecting watermarks in multi-party environment

court for copyright violation of a relation, each party having a slightly different version of the same relation. The actual situation is described in Figure 6.2. A dotted line represents a relation being distorted by a party in an attempt to destroy any watermark it contains. We assume that $\gamma < 100$ for all the parties who have watermarked the relation, which gives a high probability of the watermark being preserved if the relation is distorted or re-watermarked (Table 6.3). Hence, with a high probability, C's watermark is detected in $\tilde{R}_1$ and $\tilde{R}_2$ while d_1's watermark is detected in $\tilde{R}_3, \tilde{R}_4, \tilde{R}_5$.

In Algorithm 25, each party (including the actual owner) u_i first proves its ownership on the watermark relation it has distributed by detecting its watermark in the relation and if the watermark is detected and watermark of any other party is detected in the relation obtained from de-watermarking R, then it is clear that u_i inserted its watermark in an already watermarked relation and so, u_i cannot be the actual owner of the relation. Figure 6.1 illustrates an example where for $i = 1$ to 10, parties d_i watermark relation to obtain watermarked copy R_i. The rightful owner is d_1 who watermarked the original relation R_0 and knows about d_5, d_6, d_8 claiming ownership of R_5, R_6, R_8 respectively. These relations look very similar to R_1, so d_1 takes d_5, d_6, d_8 to court where the judge runs Algorithm 25 that identifies d_1 as the rightful owner and prosecutes d_5, d_6, d_8 for copyright violation. Experimental results confirm this as well. In the experiments, all parties except the original owner, distort the relation to a certain extent before inserting their watermarks. We used distortions varying from 20% to 40% in our experiments.

2. **Situations requiring original dataset**

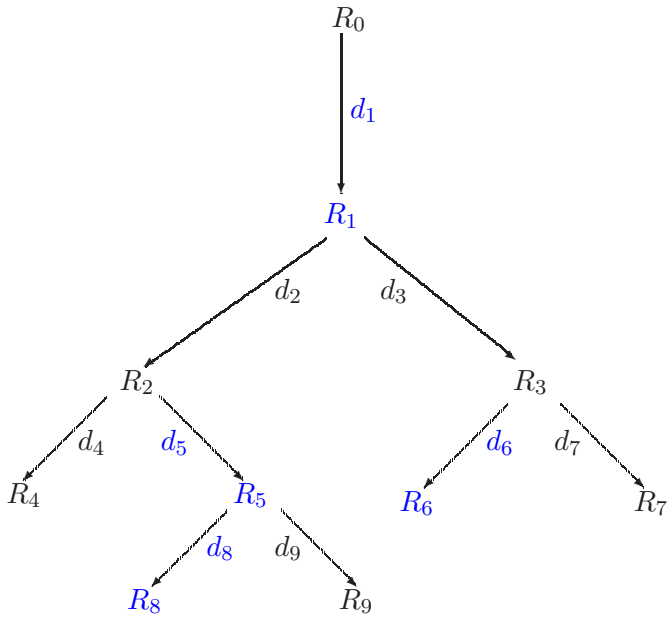

Figure 6.1: Owner identification

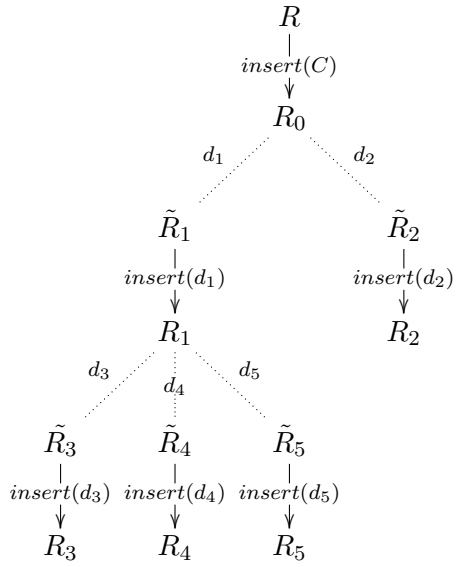

Figure 6.2: Multiple watermarking scenario - dotted lines denote distortion and solid lines denote watermarking

Often, companies require precise data where a difference of even one bit might be disastrous such as stock markets and military operations. Once a relation is watermarked in [11], it cannot be restored to it's original state if needed. Since our watermarking scheme is reversible, the original data can be restored by executing the detection algorithm. It is also possible to distribute low-quality data free of cost and users can then purchase the key to extract original data.

6.5.1 Semi-Blindness

As mentioned in 6.1, the two alternatives to facilitate reversibility are as follows.

(a) Store original bits at a secondary location before modifying (our proposed solution) or

(b) Original bits and watermark bits should be recoverable from modified bits.

There are a few ways of implementing the second option. Algorithm 23, statement 9 can be replaced by $r.A_i^j = \mathcal{H}(\mathcal{K}\|r.P) \oplus r.A_i^j \pmod 2$. But an attacker $\mathcal{A}$ can run the insertion algorithm with inputs $(R_w, K', \gamma', \upsilon', \Xi')$ and get output R', W' such that $R' \xrightarrow{ins(\mathcal{A})} R_w$. Also $R \xrightarrow{ins(\mathcal{C})} R_w$, thus making it impossible to decide who (owner/ attacker) watermarked the relation first. Thus the this solution is vulnerable to pre-image attacks.

There have been reversible watermarking algorithms, primarily for images [13, 89]. These schemes facilitate watermarking by encoding watermark bit and original value in the modified value at the cost of watermarking capacity. Another option is to use lossless compression to first compress the original bits, append watermark bits and embed resulting bitstream [22]. Since lossless compressions are sensitive to modifications, such schemes are not very resilient as suggested in [52].

The first challenge in designing a blind reversible scheme for database relations is that lossless compression technique is not resilient against attacks. The second problem is that adapting reversible image watermarking schemes is harder

because neighboring attributes or tuples do not have correlation unlike images, which is a prerequisite for schemes such as [13]. Our next research endeavor is to implement a fully blind database watermarking model by working around these two limitations.

6.6 Conclusion

The watermarking scheme proposed by Agrawal and Kiernan is irreversible, resulting in problems during owner identification in case of additive or secondary watermarking attacks. Our modified scheme is reversible and thus the rightful owner can be identified from n candidates. The major advantages of our proposed scheme are as follows,

1. It provides query preservation.
2. It identifies rightful owner if relation is watermarked by multiple parties.
3. It facilitates reversibility.

The current model requires modified bits to be stored at a secondary location ($\mathcal{ET}$). Chapter 7 eliminates this requirement and proposes a reversible blind watermarking scheme. The second enhancement is watermarking relations that do not contain a primary key. Concatenated attributes in a tuple can act as a primary key in such cases. However, the possibility of duplicate attributes makes identification of marked tuples difficult. One possibility is to treat tuples with duplicate attributes as a single tuple.

Chapter 7

Blind and Reversible Database Watermarking

7.1 Introduction

Database watermarking models are presented in previous works such as [11, 12, 86, 40, 42, 61, 62, 101, 102, 103]. A typical database watermarking scenario is when a publisher $\mathcal{C}$ creates a database relation $\mathcal{R}$ and sells it to $\mathcal{O}$. If $\mathcal{O}$ is a traitor, it illegally sells the relation to others. To prevent this, $\mathcal{C}$ embeds a watermark $\mathcal{W}$ in $\mathcal{R}$. Similarly, if a data provider $\mathcal{D}$ uploads relation $\mathcal{R}$ for remote query process, an attacker might reconstruct the original relation by assembling query results. Hence, $\mathcal{D}$ uploads a watermarked relation.

A blind watermarking scheme requires only watermarked object and a secret key to detect watermark while a non-blind watermarking scheme requires the unmarked multimedia object in addition to the first two inputs. The major disadvantage of a non-blind watermarking scheme is that one needs to store the unmarked object at a secure secondary storage location and feed it back to the detection algorithm later.

Reversible watermarking provides a mechanism to revert the watermarked relation back to the original unmarked relation using a secret key. The key advantages of reversibility are as follows.

1. It allows for trial version of multimedia content, that can be later upgraded

to the full version by reversing it. As an example, a company may want to distribute low quality (in terms of usability and precision) relations free of cost and then require customers to purchase a key that is needed to revert the relation to high quality original relation. This is impossible for irreversible watermarking schemes.

2. It permits to introduce higher distortion in the data since original data can be regenerated by reversing the watermarking.

We determine the requirements of database watermarking model, feasible attacks, and propose a reversible and blind database watermarking scheme addressing these concerns.

Several reversible and blind image watermarking schemes have been proposed. *Data compression* based reversal [22] compresses the least significant bits (LSBs) of n pixels selected into m bits, where $m < n$. These m bits and $n - m$ watermark bits are then inserted in the n selected pixels. However, data compression based watermarking schemes are extremely fragile since the lossless algorithms are not modification-resistant. *Histogram shifting* techniques [23] exploit the notion that neighboring pixels have high correlation and depending on the watermark bit, the histogram bins are circularly upgraded (if watermark=1) or downgraded (if watermark=0). Since database relation values do not possess correlation similar to images, histogram shift technique is irrelevant for our purpose. *Difference expansion* based watermarking [13, 89] integrates a watermark bit to an n-element vector such that the original vector and the watermark bit can be retrieved from the modified vector.

There are three categories of reversible image watermarking models based on data compression, difference expansion, and histogram shifting. Histogram expansion techniques [23] rely on neighboring image blocks having similar histogram values. Since this assumption does not hold for databases, we shall not consider this category any further. Data compression techniques studied in [22] work as follows:

1. Select pseudo-randomly pixels that will carry watermark. Let these pixels be from the set $\{p_1, \ldots, p_n\}$.

2. Quantify the pixels using the secret element L so the parameters are remainder $r_i = p_i - \frac{p_i}{L} * L$, quantified pixel set $p'_i = \frac{p_i}{L} * L$.

3. Compress the remainders using CALIC lossless compression algorithms from [96] to m values ($m < n$).

4. Add the m compressed remainders and $n - m$ watermark values to residues $\{p'_1, \ldots, p'_n\}$.

$$p''_i = p'_i + \begin{cases} r'_i & \text{if } i \leq m \\ w_{i-m} & \text{if } m < i \leq n \end{cases}$$

During detection, the remainders and the watermark bits are extracted and the original pixels reconstructed using decompression. Data compression based watermarking schemes are extremely fragile since the lossless algorithms are not modification-resistant.
The following example shows watermark values w_1, w_2, w_3 (integers between 0 and 7) being embedded in 9 pixels (given by the array p), where $\{r_1, \ldots, r_6\}$ are compressed remainders.

$$\text{p} = \begin{pmatrix} 34 & 45 & 37 \\ 48 & 60 & 63 \\ 39 & 72 & 57 \end{pmatrix}$$

$$\text{p'}= \begin{pmatrix} 32 & 40 & 32 \\ 48 & 56 & 56 \\ 32 & 72 & 56 \end{pmatrix}$$

$$\text{r} = \begin{pmatrix} 2 & 5 & 5 \\ 0 & 4 & 7 \\ 7 & 0 & 1 \end{pmatrix}$$

$$\text{r'} = \begin{pmatrix} r_1 & r_2 & r_3 \\ r_4 & r_5 & r_6 \\ w_1 & w_2 & w_3 \end{pmatrix}$$

$$\text{p"}= \begin{pmatrix} 32+r_1 & 40+r_2 & 32+r_3 \\ 48+r_4 & 56+r_5 & 56+r_6 \\ 32+w_1 & 72+w_2 & 56+w_3 \end{pmatrix}$$

Difference expansion based watermarking performs invertible arithmetic operations on integers. A scheme to embed $n-1$ watermark bits in n vectors is given in [13] and a specific case for $n = 2$ is described in [89] called pairwise difference expansion. For completeness, we will introduce the latter scheme here.

Given two adjacent pixels' values from a grayscale image, we compute average a and difference d using Equation 7.1.

$$a = \lfloor \frac{(x+y)}{2} \rfloor, d = x - y \tag{7.1}$$

This operation is invertible as x and y can be computed from a and d using Equation 7.2.

$$x = a + \lfloor \frac{(d+1)}{2} \rfloor, y = a - \lfloor \frac{d}{2} \rfloor \tag{7.2}$$

The integer d is now changed to $d' = 2 * d + b$ and x', y' are computed from a, d' and watermark bit b to be inserted using Equation 7.3.

$$x' = a + \lfloor \frac{(d'+1))}{2} \rfloor, y' = a - \lfloor \frac{d'}{2} \rfloor \tag{7.3}$$

The new pixel values are x', y'. One can re-calculate a, d' from x', y' using Equation 7.1. The watermark bit is simply the LSB of d' and $d = \lfloor \frac{d'}{2} \rfloor$. Now from a, d one can compute the values of x, y using Equation 3. As a working example, consider two pixels $x = 106, y = 100$. From Equation (1), $a = 103, d = 6$, assuming $b = 1$, $d' = 2 * 6 + 1 = 13$. $x' = 103 + \lfloor (13+1)/2 \rfloor = 110, y' = 103 - \lfloor (13/2) \rfloor = 97$. Hence, the new pixel values are $x' = 110, y' = 97$. At the receiver's end $a = \lfloor (110+97)/2 \rfloor = 103, d' = 110 - 97 = 13$. $b = lsb(d') = 1, d = \lfloor d'/2 \rfloor = 6$. $x = 103 + \lfloor (6+1)/2 \rfloor = 106, y = 103 - \lfloor (6/2) \rfloor = 100$. Thus we can successfully recover the watermark bit and original pixel values from the modified values.

We denote the process of reversing a pair as $\{x_r, y_r\} = Reverse\{x, y\}$.

7.2 Model of Adversary

The set of possible attacks a watermark should survive are as follows.

A1: Random bitwise flipping attacks, i.e. some bits selected at random (probably with uniform probability distribution) are modified.

A2: Subtractive attack, i.e. some tuples chosen at random are deleted.

A3: Sorting, i.e. some tuples and/or attributes are chosen at random and their positions are changed. An ordering criteria maybe chosen by the attacker and the relation is then sorted in ascending or descending order based on that criteria thereby resulting in a differently sorted relation.

A4: secondary watermarking, i.e. a watermark is superimposed on the watermarked relation.

The degree of secrecy and randomness in selecting the tuples and attributes that will be marked along with proportion of the tuples selected for marking determines the security level of watermark against the attacks **A1** and **A2**. The assumption that primary key cannot be modified by the attacker ensures that attack **A3** is not successful since the correct order can be re-established using primary key values (for example, sorting tuples in ascending order of primary key). We focus on providing security against secondary watermarking.

Assume that Alice watermarks a relation R to create watermarked relation R_a. An attacker Mallory might make some modifications in R_a before re-watermarking it with a secondary watermark to create relation R_m, Watermarks of Mallory and Alice are detected in R_m with probabilities 1 and p, respectively. So the incorrect party (Mallory) is output as the owner with probability $1 - p$, and with probability p, owner is either Mallory or Alice. The problem can be averted by designing reversible watermarking algorithms as explained below.

Considering the same situation again when Alice and Mallory both watermark a relation, when the judge needs to determine the rightful owner, he asks both Alice and Mallory to detect their watermarks in their watermarked documents R_a and R_m, respectively. They reverse their relations to the original documents R and R'_a, respectively (as Mallory might have made some modifications in R_a before

inserting the watermark). Alice's watermark is detected in the reversed relation of Mallory but Mallory's watermark is not detected in the reverse relation of Alice, which proves that the sequence of watermarking was Alice followed by Mallory and thus establishes Alice as rightful owner. Recently, a reversible scheme for database watermarking was proposed in [45]. The inserting algorithm stores the original bits that are later modified in an *embed map*. During the detection algorithm, the marked bits are sequentially replaced by bits from the *embed map*. This approach is weak in the following situations,

- the adversary deletes one of the tuples, then the bits from the tuples positioned after the deleted ones will be distorted.
- the scheme is essentially non-blind since the information about the watermark needs to be stored in a safe location.
- the database has to be updated and re-watermarked, one needs to reverse the entire relation (incremental watermarking).

Thus the main objective of our scheme is to eliminate these three shortcomings of [45] and still provide security against secondary watermarking attacks.

7.3 Proposed Scheme

The two primary objectives of our watermarking model are reversibility and blindness. Difference expansion is the most suitable method to facilitate reversibility in database watermarking since the markable data is in numeric format. In order to utilize the reversible watermarking based on difference expansion, we select two attributes A_i and A_j, from the same tuple to carry the watermark bit. We need to select the two attributes so that the distortion is within the bounds. We also need to ensure that the distortion is tolerable by checking that the change is limited to the ξ least significant bits. Let the tuple selected for watermarking be r and the attributes be A_i, A_j. The bit embedded is $lsb(\mathcal{H}(\mathcal{K}\|r.P))$. Thus, when the detection algorithm is run and bit is extracted, it is compared to $lsb(\mathcal{H}(\mathcal{K}\|r.P))$ for determining successful recovery. Since the attacker cannot modify the primary key, $lsb(\mathcal{F}(r.P))$ enables us to identify marked tuples and difference expansion facilitates reversal. The insertion and detection algorithms are provided in Algorithm 26 and

Algorithm 27 respectively.

In lines 6, 7 of Algorithm 26, we ensure that in case the unmarked attributes were reversed, the difference between the reversed values and unmarked values should exceed distortion tolerance. This condition can detect the attributes which are not marked because of exceeding distortion limits. The condition is rechecked in lines 12, 13 of Algorithm 27 once the unmarked values are computed from the marked attributes.

In the detection algorithm, we also check that a significant proportion of marks are detected in the multimedia object in order to establish beyond reasonable doubt that the object is in fact watermarked. The significance level can be determined by parameter α as in [11]. We use percentage of marks detected, *prctng*, as a simpler and equally strict significance level metric. Considering *prctng* to ensure mark presence reduces the chances of false positives if *prctng* is sufficiently large (experimental results show 85% and over is desirable).

7.4 Experimental Results

We carried out experiments with 1000 files having 200 to 300 tuples and 10 to 20 attributes each. The software generated the relations, inserted the the watermark, made modifications of the watermarked relations, and detected the watermark in the attacked files. Changing fractions did not have major effect on detectability of watermark (with the exception when fraction is equal to 33%). As tolerance increases, probability of false positives increases and probability of detection also increases. With increasing attack levels, detection probability reduces and is confirmed by the experimental results. The worst case scenario occurred when the attacker modified 48 out of every 100 tuples. In such a situation, 89 out of 100 times, the watermark was still detected corroborating the theoretical value suggested by Equation 7.5.2. Overall, 9 different fractions, 10 different attack levels, and five different tolerance levels were introduced and watermark was detected in 42167 out of 46045 watermarked files with a cumulative probability of 91.5%.

Input: relation R, private key K, fraction γ, number of markable attributes υ, LSB usage $\Xi = \{\xi_1, \xi_2, \ldots, \xi_\upsilon\}$
Output: Watermarked relation R_w

forall *tuples* $r \in R$ **do**
  if $\mathcal{F}(r.P) \pmod{\gamma} = 0$ **then**
    $i = \mathcal{F}(r.P) \pmod{\upsilon}$; `// identify attribute 1`
    $j = \mathcal{F}(\frac{r.P}{2}) \pmod{\upsilon}$; `// identify attribute 2`
    $x = max(A_i, A_j)$, $y = min(A_i, A_j)$;
    $\{x_r, y_r\} = Reverse\{x, y\}$;
    if $abs(x - x_r) > \xi_1$ *OR* $abs(y - y_r) > \xi_2$ **then**
      $a = \lfloor \frac{x+y}{2} \rfloor$, $d = x - y$;
      $b = lsb(\mathcal{H}(\mathcal{K} \| r.P))$; `// bit to embed`
      $d' = 2 * d + b$;
      $x' = a + \lfloor \frac{d'+1}{2} \rfloor$, $y' = a - \lfloor \frac{d'}{2} \rfloor$;
      if $A_i > A_j$ **then**
        $\delta_1 = abs(A_i - x')$;
        $\delta_2 = abs(A_i - y')$;
        if $\delta_1 < \xi_i$ *AND* $\delta_2 < \xi_j$ **then**
          $A_i = x'$, $A_j = y'$;
        end
      else
        $\delta_2 = abs(A_i - x')$, $\delta_1 = abs(A_i - y')$;
        if $\delta_2 < \xi_i$ *AND* $\delta_1 < \xi_j$ **then**
          $A_i = y'$, $A_j = x'$;
        end
      end
    end
  end
end

Algorithm 26: Reversible and blind watermark insertion

Input: Watermarked relation $\tilde{R}_w$, Secret parameter list $\phi = (\mathcal{K}, \gamma, \upsilon, \Xi), prcntg$
Output: {Watermark Status $\in \{true, false\}$, Restored relation R}

$R = \tilde{R}_w$, $matchcount = 0$, $totalcount = 0$;
forall *tuples* $\tilde{r}_w \in \tilde{R}_w$ **do**
  if $\mathcal{F}(r_w.P) \pmod{\gamma} = 0$ **then**
    $i = \mathcal{F}(r_w.P) \pmod{\upsilon}$; `// identify marked attribute`
    $j = \mathcal{F}(\frac{r_w.P}{2}) \pmod{\upsilon}$; `// identify marked bit`
    $x' = max(A_i, A_j)$, $y' = min(A_i, A_j)$;
    $a' = \lfloor \frac{x'+y'}{2} \rfloor$, $d' = x' - y'$;
    $b = lsb(d')$ $d = \lfloor \frac{d'}{2} \rfloor$;
    $x = a' + \lfloor \frac{d+1}{2} \rfloor$, $y = a' - \lfloor \frac{d}{2} \rfloor$;
    $\{x_r, y_r\} = Reverse\{x, y\}$;
    if $abs(x - x_r) > \xi_1$ *OR* $abs(y - y_r) > \xi_2$ **then**
      if $A_i > A_j$ **then**
        $\delta_1 = abs(A_i - x)$, $\delta_2 = abs(A_i - y)$;
        if $\delta_1 < \xi_i$ *AND* $\delta_2 < \xi_j$ **then**
          if $b = lsb(\mathcal{H}(\mathcal{K} \| r.P))$ **then**
            $A_i = x$, $A_j = y$;
            $matchcount = matchcount + 1$;
          end
          $totalcount = totalcount + 1$;
        end
      else
        $\delta_2 = abs(A_i - x)$, $\delta_1 = abs(A_i - y)$;
        if $\delta_2 < \xi_i$ *AND* $\delta_1 < \xi_j$ **then**
          if $b = lsb(\mathcal{H}(\mathcal{K} \| r.P))$ **then**
            $A_i = y$, $A_j = x$;
            $matchcount = matchcount + 1$;
          end
          $totalcount = totalcount + 1$;
        end
      end
    end
  end
end
if $\frac{matchcount}{totalcount} \geq prctng$ **then**
  return $\{true, R\}$;
else
  return $\{false, \tilde{R}_w\}$;
end

Algorithm 27: Reversible and blind watermark detection

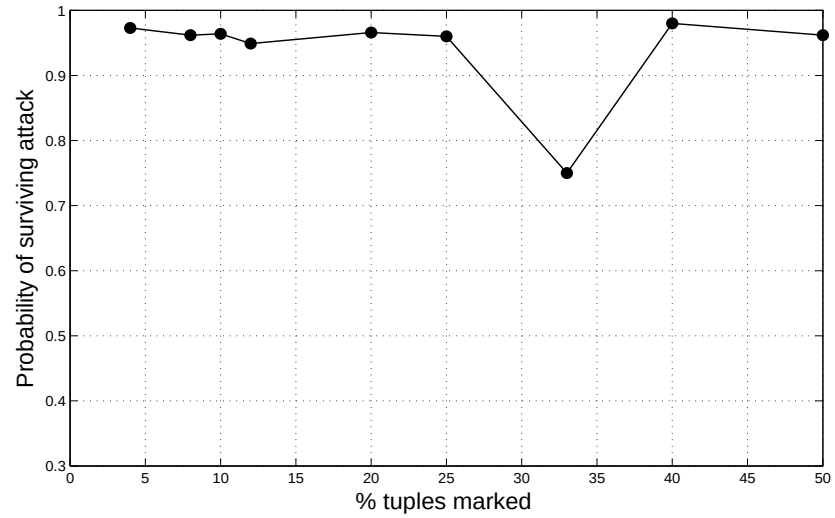

Figure 7.1: Effect of changing fraction of tuples marked on detection

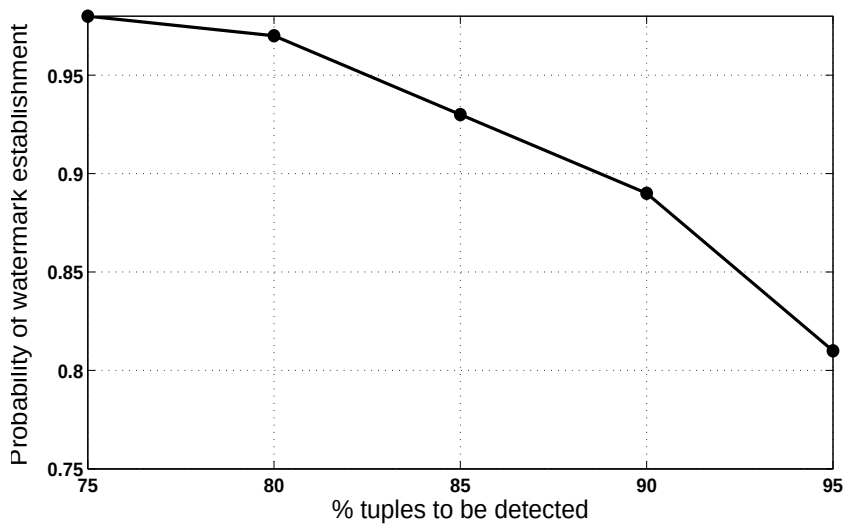

Figure 7.2: Effect of changing percentage of marks that need to be detected to establish watermark presence

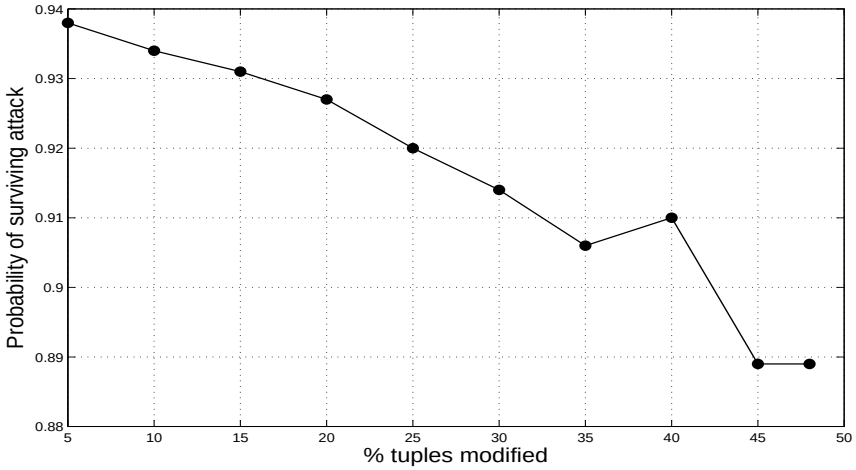

Figure 7.3: Effect of changing attack levels on detection

7.5 Analysis

We shall now analyze the capacity and security properties of the watermarking scheme as compared to previous schemes such as [11, 45] and the advantages our schemes proposes over the previous schemes.

7.5.1 Capacity

In our scheme, γ tuples out of every 100 tuples are selected for watermarking. Thus the capacity of our scheme is given by $\mathcal{C} = \frac{\eta}{\gamma}$, where η is the total number of tuples. The capacity is, theoretically, the same as the capacity of previous scheme mentioned in [11]. Distortion levels Ξ used in our schemes are much higher. The modified values can later be reversed back to original values upon purchase of full version of the data set. Allowing higher distortion results in more attributes selected for marking actually getting marked thereby increasing the capacity in practice.

7.5.2 Security

In terms of security, the possible attacks to consider are given in Section 7.2. Next we discuss the security of our proposed solution.

Random bitwise flipping attack. If we assume that the attacker has complete knowledge of Ξ and υ. The attacker can now choose randomly tuples ζ and flip all the ξ_i LSBs of attribute A_i $(1 \leq i \leq \upsilon)$ in those tuples. This attack is successful if the attacker can toggle sufficient marked bits such that detection algorithm detects less than τ watermarked bits correctly. Hence the attacks succeeds only when attacker modifies at least $\omega - \tau + 1$ watermarked bits, where $\omega = \frac{\eta}{\gamma}$ is the total number of tuples marked. The probability of this attack is given by Equation 7.5.2 (see [11]). This probability is the same as [11, 45]. For $\gamma = 50$, the worst case scenario is when attacker changes 48% of the tuples and the success probability of attack is merely 11% as confirmed by experiments and shown in Figure 7.3. If the attacker changes more than half the tuples, a) the usability would be assumed to be severely affected, and, b) watermark would be detected in the bitwise complemented relation.

$$\mathcal{P}(\mathcal{A}) = \sum_{i=\bar{\tau}}^{\omega} \frac{\begin{pmatrix} \omega \\ i \end{pmatrix} \begin{pmatrix} \eta - \omega \\ \zeta - i \end{pmatrix}}{\begin{pmatrix} \eta \\ \zeta \end{pmatrix}} \tag{7.4}$$

Subtractive attack: This type of attack is similar to the previous attack in that the attacker has to again remove at least $\omega - \tau + 1$ marked tuples out of η tuples such that the detection algorithm detects less than τ matches. The probability of this attack is same as the previous attack (random bitwise flipping attack).

Sorting: If an attacker re-sorts the tuples based on any attribute, it does not effect the detection algorithm. Since the watermark detection is carried out of each tuple independently, any change in order does not effect the outcome of the detection algorithm. Sorting attack was given significant importance while deciding difference expansion method to be used.

Secondary watermarking: Let us consider a situation where Alice watermarks relation R resulting in relation R_a ($R \xrightarrow{ins(Alice)} R_a$) and distributes it for trial. The attacker Mallory modifies R_a to R'_a and re-watermarks R'_a resulting in relation R_m. R'_a still contains Alice's watermark with a high probability p and Alice's

watermark is successfully removed by Mallory with a probability $1 - p$ (According to experimental results, $p \approx 0.89$ for $\gamma = 50$). R_m contains Mallory's watermark with probability 1 since it has not been modified after watermark insertion. Let R_a accidently contain Mallory's watermark with a negligible probability δ ($\delta \approx 0$).

The judge asks Alice and Mallory to run detection algorithm on R_a and R_m respectively. Both Mallory's and Alice's watermarks are successfully detected in their respective relations. Mallory's restored relation is R'_a and Alice's restored relation is R. With a high probability p, Alice's watermark is detected in R'_a but Mallory's watermark is detected with an extremely low probability δ in R. Thus it becomes evident that Mallory inserted the watermark in the relation already watermarked by Alice and thereby Alice is the rightful owner. In this way, the current watermarking scheme defeats secondary watermarking attacks.

The watermarking scheme identifies the correct owner if more than two parties insert their watermarks in the relation. A modified version of Algorithm 25 is executed with the only difference that we do not require the *embed map* to detect the watermark. Algorithm 28 provides the modified procedure.

Input: Potential owners $\mathcal{U} = \{u_1, u_2, \ldots, u_n\}$. Secret parameter list of each u_i, $\mathcal{I}_{u_i} = \{\mathcal{K}_i, \gamma_i, \upsilon_i, \Xi_i\}$, tolerance ϵ, Potential owners' versions of the watermarked relation $\{R_1, R_2, \ldots, R_n\}$
Output: Owner O

```
forall u_i ∈ U do
    if detect(R_i, I_{u_i}) == {false, R'_i} then
        U = U \ u_i;
    end
    if detect(R_i, I_{u_i}) == {true, R_i^rev} then
        if {u_j : detect(R_i^rev, u_i) == {true, R_temp}, ∀j ≠ i} ≠ null then
            U = U \ u_i;
        end
    end
    return U;
end
```

Algorithm 28: Blind owner identification

7.6 Conclusion

We have proposed a reversible and blind database watermarking model. The maximum distortion introduced to the attributes is limited to the tolerance parameter Ξ. It is, in practice, desirable to have distortion on the higher side since the watermarking is reversible. The distorted database is available to everyone and the accurate database can be purchased upon payment by users by reversing the watermarking. The proposed scheme is successful in achieving the major objective of eliminating the shortcomings of irreversible schemes like [11] mentioned in Section 6.1. The capacity of the proposed watermarking scheme is high and the attack resistance probability between 89 and 98 percent. Our future research is directed towards increasing the watermark carrying capacity and level of attack resistance in a reversible and blind watermarking model.

Chapter 8

Conclusion and future research

8.1 Thesis Summary

During this research, we focussed on developing efficient constraint-based watermarking techniques for digital multimedia objects belonging to natural language documents, software and databases. During the course of this PhD, several watermarking requirements such as robustness, blindness and reversibility, were identified. These requirements were incorporated in the subsequent models of watermarking. With greater understanding of the requirements, we were able to construct more comprehensive and effective watermarking schemes. The improvements included localization of attack to distorted section in natural language documents, resilience against automated debugger-based attacks in software codes, and security against secondary watermarking attacks through reversibility in database watermarking. We have analyzed our proposed watermarking schemes in terms of security and capacity. We demonstrated that our watermarking schemes are secure against the various possible attacks while having low probability of false positives and have sufficient watermark-carrying capacity to establish ownership. We implemented our schemes using C (Natural language watermarking), C++ (Software and database watermarking) and Java (AES permutations for natural language watermarking) languages on Windows platform.

We addressed three categories of multimedia objects (in chronological order),

1. *Natural language watermarking*: Identifying the three approaches to watermarking text (format-based, synonymy-based and semantic-based) was the first task in this project. It became evident that the first two categories are

not very robust against generic attacks, and that the watermark should be contained in the *meaning* of the text rather than the *formatting* or *wording*. We proposed a format-based natural language watermarking scheme that addressed robustness, capacity imperceptibility requirements, also incorporating error-correction codes for higher resilience against attacks.

2. *Software watermarking*: Several approaches are adopted to watermark software source code, amongst which, the more effective methods are register allocation, class obfuscation and branch modifications. While register allocation is susceptible to automated attacks, and class obfuscation is applicable to only object-oriented programming languages (and in that too, violates fundamental programming principles), branch modification is a robust way of watermarking source codes. It is also applicable to a wider, more diverse range of the programming languages. The wide application of branching is because it is a fundamental operation in programming and almost all languages support branching in some form. We analyzed a branch based watermarking scheme proposed in [71] that converts jump instructions to function calls such that the added function transfers the control to the correct destination, generating the watermark as well. We identified a crucial loophole that can be exploited by an attacker to identify the fingerprint branch function in an automated manner and also re-calculate the original target of jump instruction thereby destroying the watermark. This loophole was fixed in our proposed scheme. The new watermarking scheme is resilient against automated attacks and needs extensive manual inspection of the program to identify and eliminate the fingerprint branch function.

3. *Database watermarking*: The current database schemes are mostly modified versions of [11, 12] by Agrawal and Kiernan. The strength of the scheme lies in its simplicity; the tuples are independently watermarked under the condition that primary key cannot be modified by the attacker and therefore, tuples that carry watermark bits can symmetrically be identified at insertion and detection steps. This scheme offers high strength against additive, subtractive and bit-flipping attacks but does not sufficiently address secondary watermarking attacks. An attacker's watermark may destroy the original author's watermark. To reduce this possibility, we proposed facilitating reversibility so that the original relation can be regenerated from the watermarked relation. Given

that the watermarking is done reversibly, the original author can be identified through back-tracking as described in Sections 7.5.2, 6.5. Using this approach, in conjunction with the base watermarking scheme of [11, 12], we can achieve a robust watermarking model that is resilient against secondary watermarking attacks. We have presented two reversible database watermarking schemes accompanied with a semi-blind scheme requiring an *embed map* in addition to the watermarked relation and secret key to reverse the relation and detect the watermark and the second eliminating this constraint to provide a *blind* watermarking that requires just the watermarked relation and secret key to detect the watermark.

Another aspect of watermarking schemes that we identified is the environment in which the watermarking schemes should be implemented. Understanding this environment is critical in order to fully realize the watermarking potential. For example, in a peer-to-peer multimedia distribution environment, the publishers must direct their attention towards either the uploaders of digital content or the downloaders. This provides a focussed approach towards building a watermarking model.

Study of different types of multimedia objects showed us the constraints that apply to each, and how these constraints directly result in boundaries for watermarking schemes. For example, attributes should not be modified in databases beyond a certain limit unless the watermarking is reversible otherwise the queries on the database might return distorted results. Similarly, the natural language domain has grammatical constraints that limit the watermark-carrying capacity of the documents. In software watermarking, we need to preserve the software interference graph for typical user inputs. Hence, the watermark component in a watermarked software should not interfere with its functionality unless a specific input is given to the program to detect watermark.

A practical issue identified with watermarking environment is the ability to discourage illegal distribution using bogus media uploading. Multimedia objects on a peer-to-peer network usually originate from a single uploader from whom multiple primary entities download the multimedia object. These primary entities may then choose to themselves upload the media or exclude themselves from further distribution. From publishers' point of view (such as Universal Studios or Warner Brothers), if one or more fake multimedia copies are uploaded, then it would lead primary entities into downloading *garbage* since nobody can verify the data. By the time the

primary entities realize this, they already waste a significant amount of bandwidth and virtually nobody wants to be the primary downloader since the uploaded copies are not verified. Publishers are already uploading bogus copies of their multimedia objects. This is a very effective scheme against illegal content distribution online.

Characteristics of standard watermarking scheme are robustness, blindness, detectability, high watermark-carrying capacity, and low false positives. In addition to these characteristics, we propose another feature that a watermarking scheme should satisfy *reversibility*. Reversibility is critical because the original multimedia object can be re-generated from the watermarked copy and therefore, it also provides greater watermark-carrying capacity by placing lesser constraints. At the same time, reversible watermarking provides a solution against secondary watermarking attacks.

8.2 Future Research Directions

The watermark community views reversibility as a desirable feature of a watermarking scheme, but not classify it as an essential characteristic. This is because reversibility introduces several challenges when designing a watermarking model, such as ability to deterministically identify marked tuples versus rejected tuples. However, reversibility should be given greater importance in the process of designing a watermarking scheme and researchers should strive to make their watermarking models reversible. We believe that reversibility should be included as a core requirement in future watermarking model proposals, and be met rigorously in order to establish effectiveness of the watermarking scheme. Also, the reversibility requirement is independent of the multimedia object that is being watermarked. Reversibility can be achieved through different methods including signal transformations and arithmetic operations. Reversibility can sometimes be easier to achieve in some categories of multimedia objects such as images and databases than in other categories like natural language documents.

So far in the field of copyright protection, watermarking has been the focus of study, primarily due to the relative ease of implementation as compared to fingerprinting. But with the popularity of peer-to-peer technology, it is important to insert fingerprints in addition to the watermarks so that the primary source of such distribution can also be identified. To facilitate this, research should be carried out in constructing robust and short collusion-secure codes like the one proposed in [16].

In terms of extending watermarking schemes to novel multimedia objects, HTML and XML documents, DNA sequences, numeric sets and graphs, digital maps, statistical results and emails should also be considered. Each multimedia object has its own set of watermark-carrying constraints, and simultaneously offers some kind of watermarking channel that can be exploited. XML documents, especially, are of interest given the increasing popularity of these types of database repositories. Since the data inside these documents can directly be used and transformed into other forms, such as address books and list of customers, this opens avenues for commercial applications, thereby making watermarking of XML documents useful and lucrative.

Bibliography

[1] *Link Parser*. AbiWord, available at http://www.abisource.com/downloads/link-grammar/4.3.5/link-grammar-4.3.5.tar.gz.

[2] *MSDN Visual C++ Language Reference*. Microsoft, available at http://msdn.microsoft.com/en-us/library/45yd4tzz(VS.80).aspx.

[3] *The Parts of Speech*. University of Ottawa, available at http://www.arts.uottawa.ca/writcent/hypergrammar/partsp.html.

[4] *English Grammar Key Words, Terms, Definitions*. EnglishGuide, available at http://englishguide.us/key-terms.

[5] *Online Dictionary, Encyclopedia and Thesauru*. available at http://www.thefreedictionary.com/.

[6] *iscribe*. available at http://www.iscribe.org/english/def.html.

[7] L. Sue Baugh. *Essentials of English Grammar*. McGraw-Hill, 2^{nd} edition, 1993.

[8] David Crystal. *The Cambridge Encyclopedia of Language* . Cambridge University Press, 2^{nd} edition, 1997.

[9] *True Spanish documents*. National Park Services, available at http://www.nps.gov/archive/tuma/TrueDocs.html.

[10] *World currencies*. Corpus Christi Coin and Currency Inc., available at http://www.cccoin1.com/worldcurrency.htm

[11] R. Agrawal and J. Kiernan. Watermarking relational databases. In *Proceedings of the 28th International Conference on Very Large Databases VLDB*, 2002.

[12] Rakesh Agrawal, Peter J. Haas, and Jerry Kiernan. Watermarking relational data: framework, algorithms and analysis. *The VLDB Journal*, 12(2):157–169, 2003.

[13] A.M. Alattar. Reversible watermark using the difference expansion of a generalized integer transform. *IEEE Transactions on Image Processing*, 13(8):1147–1156, 2004.

[14] M.J. Atallah, V. Raskin, M. Crogan, C. Hempelmann, F. Kerschbaum, D. Mohamed, and S. Naik. Natural language watermarking: design, analysis, and a proof-of-concept implementation. In *Proceedings of 4th Information Hiding Workshop, LNCS*, pages 185–199, Pittsburgh, Pennsylvania, 2001. Springer-Verlag, Heidelberg.

[15] Igor A. Bolshakov. A method of linguistic steganography based on collocationally-verified synonymy. In *In Proceedings of 4th International Workshop on Information Hiding, IH 2004*, volume 3200 of *LNCS*, pages 180–191. Springer Verlag, 2005.

[16] Dan Boneh and James Shaw. Collusion-secure fingerprinting for digital data. *Advances in Cryptology CRYPT0 95, Lecture Notes in Computer Science*, 963:452 – 465, 1995.

[17] Grady Booch. *Object oriented design with applications.* Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1991.

[18] J. Brassil, S. Low, N. F. Maxemchuk, and L. O'Gorman. Hiding information in documents images. In *Proceedings of Conference on Information Sciences and Systems (CISS-95)*, 1995.

[19] J.T. Brassil, S. Low, N.F. Maxemchuk, and L. O'Gorman. Marking text features of document images to deter illicit dissemination. In *Proceedings of the 12th IAPR International Conference on Computer Vision and Image Processing*, volume 2, pages 315 – 319, Jerusalem, Israel, October 1994.

[20] Charles Briquet. *Les filigranes.* Publisher: Hacker Art Books; Facsimile Ed edition (June 1985), 1907.

[21] Leslie Cauley. *U.S. Net access not all that speedy.* USA today, available at http://www.usatoday.com/tech/news/techpolicy/2007-06-25-net-speeds_N.htm

[22] Sharma G.-Tekalp M.A. Saber-E. Celik, M.U. Reversible data hiding. In *Proceedings of International Conference on Image Processing*, volume 2, pages 157–160, September 2002.

[23] Chin-Chen Chang, Wei-Liang Tai, and Min-Hui Lin. A reversible data hiding scheme with modified side match vector quantization. In *AINA '05: Proceedings of the 19th International Conference on Advanced Information Networking and Applications*, pages 947–952, Washington, DC, USA, 2005. IEEE Computer Society.

[24] N. Chotikakamthorn. Electronic document data hiding technique using inter-character space. In *Proceedings of The 1998 IEEE Asia-Pacific Conference on Circuits and Systems, IEEE APCCAS 1998*, pages 419–422, Chiangmai, Thailand, November 1998.

[25] N. Chotikakamthorn. Document image data hiding technique using character spacing width sequence coding. In *Proceedings of International Conference on Image Processing, ICIP 1999*, volume 2, pages 250–254, Kobe, Japan, October 1999.

[26] Christian Collberg, Edward Carter, Saumya Debray, Andrew Huntwork, Cullen Linn, and Mike Stepp. Dynamic path-based software watermarking. In *Proceedings of Conference on Programming Language Design and Implementation*, volume 39, pages 107–118, June 2004.

[27] Christian Collberg, Andrew Huntwork, Edward Carter, and Gregg Townsend. Graph theoretic software watermarks: Implementation, analysis, and attacks. In *Proceedings of 6th Information Hiding Workshop, LNCS*, volume 3200, pages 192–207, 2004.

[28] Christian Collberg, Stephen Kobourov, Edward Carter, and Clark Thomborson. Error-correcting graphs for software watermarking. In *Proceedings of 29th Workshop on Graph Theoretic Concepts in Computer Science*, pages 156–167, 2003.

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

[30] Christian S. Collberg and Clark Thomborson. Watermarking, tamper-proofing, and obfuscation - tools for software protection. In *IEEE Transactions on Software Engineering*, volume 28, pages 735–746, August 2002.

[31] Joan Daemen and Vincent Rijmen. The block cipher rijndael. In *CARDIS '98: Proceedings of the The International Conference on Smart Card Research and Applications*, pages 277–284, London, UK, 2000. Springer-Verlag.

[32] Joan Daemen and Vincent Rijmen. *ADVANCED ENCRYPTION STANDARD (AES)*. available at http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf, 2001.

[33] C. J. Date. *Introduction to Database Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[34] Vjekoslav Dorn. A contribution to the history of spectacles in croatia. *Documenta Ophthalmologica*, 86(2):173–189, 1994.

[35] Ramez A. Elmasri and Shankrant B. Navathe. *Fundamentals of Database Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[36] Zhihao Zhang Fei Guo, Jianmin Wang and Deyi Li. A new scheme to fingerprint xml data. In *In Proceedings of Workshop on Knoledge Discovery from XML Documents, KDXD 2006*, volume 3915 of *LNCS*, pages 85–94. Springer Verlag, 2006.

[37] Alice E. Fischer and Frances S. Grodzinsky. *The anatomy of programming languages*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[38] Daniel P. Friedman, Christopher T. Haynes, and Mitchell Wand. *Essentials of programming languages (2nd ed.)*. Massachusetts Institute of Technology, Cambridge, MA, USA, 2001.

[39] Kazuhide Fukushima and Kouichi Sakurai. A software fingerprinting scheme for java using classfiles obfuscation. In *Proceedings of Information Security Applications, LNCS*, volume 2908, pages 303–316, 2004.

[40] David Gross-Amblard. Query-preserving watermarking of relational databases and xml documents. In *Proceedings of the 20th ACM Symposium on Principles of Database Systems*, pages 191–201, June 2003.

[41] Carl A. Gunter. *Semantics of Programming Languages: Structures and Techniques.* Foundations of Computing. MIT Press, 1992.

[42] Fei Guo, Jianmin Wang, and Deyi Li. Fingerprinting relational databases. In *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*, pages 487–492, New York, NY, USA, 2006. ACM Press.

[43] Gaurav Gupta and Josef Pieprzyk. A low-cost attack on branch-based software watermarking schemes. In *Proceedings of Fifth International Workshop on Digital Watermarking (IWDW)*, pages 282–293, 2006.

[44] Gaurav Gupta and Josef Pieprzyk. Reversible and semi-blind relational database watermarking. In *Proceedings of Second International Conference on Signal Processing and Multimedia Applications (SIGMAP)*, pages 283–290, 2007.

[45] Gaurav Gupta and Josef Pieprzyk. Reversible and semi-blind relational database watermarking. In *Proceedings of International Conference on Signal Processing and Multimedia Applications*, July 2007.

[46] Gaurav Gupta and Josef Pieprzyk. Software watermarking resilient to debugging attacks. *Journal of Multimedia*, 2(2):10–16, 2007.

[47] Gaurav Gupta and Josef Pieprzyk. Reversible and blind database watermarking using difference expansion. In *Proceedings of eForensics*, pages 1–6, January 2008.

[48] Gaurav Gupta and Josef Pieprzyk. Source code watermarking based on function dependency oriented sequencing. In *Proceedings of Fourth International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIHMSP)*, pages 282–293, 2008.

[49] Gaurav Gupta, Josef Pieprzyk, and Hua Xiong Wang. An attack-localizing watermarking scheme for natural language documents. In *Proceedings of the 2006 ACM Symposium on Information, computer and communications security (ASIACCS)*, pages 157–165, New York, NY, USA, 2006. ACM.

[50] D. Huang and Y. Hong. Inter-word distance changes represented by sine waves for watermarking text images. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(12):1237–1245, December 2001.

[51] Darek Michalek Jarek Pastuszak and Josef Pieprzyk. Copyright protection of object-oriented software. In *Information Security and Cryptology - ICISC 2001: 4th International Conference*, pages 186–199, Seoul, Korea, December 6-7, 2001.

[52] Chwei-Shyong Tsai Yen-Ping Chu Jen-Bang Feng, Iuon-Chang Lin. Reversible watermarking: Current status and key issues. *International Journal of Network Security*, 2(3):161–171, May 2006.

[53] H.P. Ji, J.E. Sook, and H. Young. A new digital watermarking for text document images using diagonal profile. In *Proceedings of Second IEEE Pacific Rim Conference on Multimedia, PCM 2001. LNCS*, volume 2195, pages 748–755, Beijing, China, October 2001. Springer-Verlag, Heidelberg.

[54] H.P. Ji, J.E. Sook, and H. Young. A new digital watermarking for text document images using diagonal profile. In *Proceedings of Second IEEE Pacific Rim Conference on Multimedia, PCM 2001. LNCS*, volume 2195, pages 748–755, Beijing, China, October 2001. Springer-Verlag, Heidelberg.

[55] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition*. Prentice Hall, January 2000.

[56] Mohan S. Kankanhalli and K. F. Hau. Watermarking of electronic text documents. *Electronic Commerce Research*, 2(1-2):169–187, 2002.

[57] Farinaz Koushanfar, Gang Qu, and Miodrag Potkonjak. Intellectual property metering. In *IHW '01: Proceedings of the 4th International Workshop on Information Hiding*, pages 81–95, London, UK, 2001. Springer-Verlag.

[58] Julien Lafaye and David Gross-Amblard. Xml streams watermarking. In *In Proceedings of Data and Applications Security, 2006*, volume 4127 of *LNCS*, pages 74–88. Springer Verlag, 2006.

[59] Raymond Lewallen. *4 major principles of Object-Oriented Programming*, available at http://codebetter.com/blogs/raymond.lewallen/pages/59908.aspx.

[60] Qiming Li and Ee-Chien Chang. On the possibility of non-invertible watermarking schemes. In *In Proceedings of 6th International Workshop on Information Hiding, IH 2006*, volume 4437 of *LNCS*. Springer Verlag, 2004.

[61] Yingjiu Li and Robert Huijie Deng. Publicly verifiable ownership protection for relational databases. In *Proceedings of the 2006 ACM Symposium on Information, computer and communications security, ASIACCS 2006*, pages 78–89, Taipei, Taiwan, 2006. ACM.

[62] Yingjiu Li, Huiping Guo, and Sushil Jajodia. Tamper detection and localization for categorical data using fragile watermarks. In *DRM '04: Proceedings of the 4th ACM workshop on Digital rights management*, pages 73–82, New York, NY, USA, 2004. ACM Press.

[63] Yingjiu Li, Vipin Swarup, and Sushil Jajodia. Fingerprinting relational databases: Schemes and specialties. *IEEE Transactions on Dependable and Secure Computing*, 02(1):34–45, 2005.

[64] Yanmei Fang Limin Gu and Jiwu Huang. Revaluation of error correcting coding in watermarking channel. In *In Proceedings of Chinese American Networking Symposium, CANS 2005*, volume 3810 of *LNCS*, pages 274–287. Springer Verlag, 2005.

[65] S.H. Low, N.F. Maxemchuk, J.T. Brassil, and L. O'Gorman. Document marking and identification using both line and word shifting. In *Proceedings of 14th Annual Joint Conference of the IEEE Computer and Communications Societies. Bringing Information to People, INFOCOM 1995*, volume 2, pages 853–860, Boston, USA, April 1995.

[66] Christopher D. Manning and Hinrich Schtze. *Foundations of Statistical Natural Language Processing*. The MIT Press, June 1999.

[67] N. Maxemchuk and S. Low. Marking text documents. In *Proceedings of International Conference on Image Processing*, pages 13–20, Washington, USA, October 1997.

[68] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1996.

[69] Akito Monden, Hajimu Iida, Ken ichi Matsumoto, Koji Torii, and Katsuro Inoue. A practical method for watermarking java programs. In *Proceedings of 24th International Computer Software and Applications Conference (COMPSAC '00)*, pages 191–197, Washington, DC, USA, 2000. IEEE Computer Society.

[70] Ginger Myles and Christian Collberg. Software watermarking through register allocation: Implementation, analysis, and attacks. In *Proceedings of International Conference on Information Security and Cryptology, LNCS*, volume 2971, pages 274– 293, 2003.

[71] Ginger Myles and Hongxia Jin. Self-validating branch-based software watermarking. In *Proceedings of 7th Information Hiding Workshop, LNCS*, volume 3727, pages 342–356, 2005.

[72] Jasvir Nagra and Clark Thomborson. Threading software watermarks. In *Proceedings of 6th Information Hiding Workshop, LNCS*, volume 3200, pages 208–223, 2004.

[73] Wilfred Ng and Ho-Lam Lau. Effective approaches for watermarking xml data. In *In Proceedings of 10th International Conference on Database Systems for Advanced Applications, DASFAA 2005*, volume 3453 of *LNCS*, pages 68–80. Springer Verlag, 2005.

[74] Josef Pieprzyk. Fingerprints for copyright software protection. In *Proceedings of 3rd Information Hiding Workshop, LNCS*, pages 178–190, 1999.

[75] Gang Qu and Miodrag Potkonjak. Analysis of watermarking techniques for graph coloring problem. In *Proceedings of International Conference on Computer Aided Design*, pages 190–193, 1998.

[76] Gang Qu and Miodrag Potkonjak. Hiding signatures in graph coloring solutions. In *Proceedings of 3rd Information Hiding Workshop, LNCS*, volume 1768, pages 348–367, 1999.

[77] Gang Qu and Miodrag Potkonjak. Fingerprinting intellectual property using constraint-addition. In *DAC '00: Proceedings of the 37th conference on Design automation*, pages 587–592, New York, NY, USA, 2000. ACM.

[78] Gang Qu, Jennifer L. Wong, and Miodrag Potkonjak. Optimization-intensive watermarking techniques for decision problems. In *DAC '99: Proceedings of the 36th ACM/IEEE conference on Design automation*, pages 33–36, New York, NY, USA, 1999. ACM.

[79] Gang Qu, Jennifer L. Wong, and Miodrag Potkonjak. Fair watermarking techniques. In *ASP-DAC '00: Proceedings of the 2000 conference on Asia South Pacific design automation*, pages 55–60, New York, NY, USA, 2000. ACM.

[80] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 26(1):96–99, 1983.

[81] Bruce Schneier. *Applied Cryptography.* John Wiley & Sons, 1996.

[82] Robert W. Sebesta. *Concepts of Programming Languages.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1998.

[83] C. E. Shannon. A mathematical theory of communication. *Bell System Technology Journal,*, 27:379–423; 623–656, 1948.

[84] Ichiro Murase, Osamu Takizawa, Tustomu Matsumoto, Hiroshi Nakagawa, Shingo Inoue and Kyoko Makino. A proposal on information hiding methods using XML. In *In Proceedings of 1st NLP and XML Workshop*, November 2001.

[85] R. Sion, M.J. Atallah, and S. Prabhakar. On watermarking numeric sets. In *1st International Workshop on Digital Watermarking*, volume 2163, pages 130–146, Seoul, Korea, November 2002. Springer-Verlag, Heidelberg.

[86] Radu Sion, Mikhail Atallah, and Sunil Prabhakar. Rights protection for relational data. *IEEE Transactions on Knowledge and Data Engineering*, 16(12):1509–1525, December 2004.

[87] Mikhail Sosonkin, Gleb Naumovich, and Nasir Memon. Obfuscation of design intent in object-oriented applications. In *Proceedings of 3rd ACM workshop on Digital Rights Management*, pages 142–153, 2003.

[88] Clark Thomborson, Jasvir Nagra, Ram Somaraju, and Charles He. Tamper-proofing software watermarks. In *Proceedings of Australasian Information Security Workshop*, volume 32, pages 27– 36, 2004.

[89] Jun Tian. Reversible data embedding using a difference expansion. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(8):890–896, 2003.

[90] Ramarathnam Venkatesan, Vijay Vazirani, and Saurabh Sinha. A graph theoretic approach to software watermarking. In *Proceedings of 4th Information Hiding Workshop, LNCS*, volume 2137, pages 157–168, 2001.

[91] Johann Gotthelf Fischer von Waldheim. *Beschreibung einiger typographischen Seltenheiten.* Google (Digitized version), 1804.

[92] Misty Wilson. *All American: Language: Syntax.* University of Northen Carolina, available at http://www.uncp.edu/home/canada/work/markport/language/grammar/spg2001/

[93] Andrea Wise, Kassandra Coghlan, and Bill Hamilton. *Whistler's watermarks.* National Gallery of Australia, available at http://www.nga.gov.au/conservation/Watermarks/listing.cfm.

[94] M. Wolf. Covert channels in lan protocols. In *Workshop on Local Area Network Security*, volume 396, pages 91–101, Karlsruhe, Germany, April 1989. Springer-Verlag, Heidelberg.

[95] Greg Wolfe, Jennifer L. Wong, and Miodrag Potkonjak. Watermarking graph partitioning solutions. In *DAC '01: Proceedings of the 38th conference on Design automation*, pages 486–489, New York, NY, USA, 2001. ACM.

[96] Xiaolin Wu. Lossless Compression of Continuous-Tone Images via Context Selection, Quantization, and Modeling. *IEEE Transactions on Image Proceesing*, 6(5):656–664, 2007.

[97] Xiangrong Xiao, Xingming Sun, and Minggang Chen. Second-LSB dependent robust watermarking for relational database. In *IAS '07: Proceedings of the Third International Symposium on Information Assurance and Security*, pages 292–300, Washington, DC, USA, 2007. IEEE Computer Society.

[98] Il-Seok Oh Young-Won Kim, Kyung-Ae Moon. A text watermarking algorithm based on word classification and inter-word space statistics. In *Proceedings of Conference on Document Analysis and Recognition (ICDAR03)*, 1995.

[99] Chuanxiang Ma Yu Fu, Cong Jin. A novel relational database watermarking algorithm. In *Proceedings of Pacific Asia Workshop on Intelligence and Security Informatics, PAISI, 2007, LNCS*, April 2007.

[100] Wen-Tai Hsieh Yuei-Lin Chiang, Lu-Ping Chang and Wen-Chih Chen. Natural language watermarking using semantic substitution for chinese text. In *Proceedings of 2nd International Workshop on Digital Watermarking, IWDW 2002. LNCS*, volume 2939, pages 129–140, Seoul, Korea, October 2003. Springer-Verlag, Heidelberg.

[101] Yong Zhang, Xia-Mu Niu, and Dongning Zhao. A method of protecting relational databases copyright with cloud watermark. *Transactions of Engineering, Computing and Technology*, 3:170–174, 2004.

[102] Yong Zhang, Bian Yang, and Xia-Mu Niu. Reversible watermarking for relational database authentication. *Journal of Computers*, 17(2):59–66, 2006.

[103] Z.H. Zhang, X.M. Jin, J.M. Wang, and D.Y. Li. Watermarking relational database using image. In *Proceedings of 3rd International Conference on Machine Learning and Cybernetics*, volume 3, pages 1739–1744, August 2004.

[104] Xuan Zhou, HweeHwa Pang, Kian-Lee Tan, and Dhruv Mangla. Wmxml: a system for watermarking xml data. In *In Proceedings of the 31st International Conference on Very large data bases, VLDB 2005*, pages 1318–1321. VLDB Endowment, 2005.

[105] William Zhu. Informed recognition in software watermarking. In *Proceedings of Pacific Asia Workshop on Intelligence and Security Informatics, PAISI, 2007, LNCS*, April 2007.

[106] William Zhu and Clark Thomborson. Algorithms to watermark software through register allocation. In *Proceedings of Digital Rights Management: Technology, Issues, Challenges and Systems, DRMTICS, 2005, LNCS*, October 2005.

[107] William Zhu and Clark Thomborson. Extraction in software watermarking. In *Proceedings of the 8th workshop on Multimedia and security, MM&SEC*, pages 175–181, New York, NY, USA, 2006. ACM.

[108] William Zhu and Clark Thomborson. Recognition in software watermarking. In *MCPS '06: Proceedings of the 4th ACM international workshop on Contents protection and security*, pages 29–36, New York, NY, USA, 2006. ACM.

[109] Umut Topkara and Mercan Topkara and Mikhail J. Atallah. The hiding virtues of ambiguity: quantifiably resilient watermarking of natural language text through synonym substitutions. In *Proceedings of MM&Sec '06: Proceedings of the 8th workshop on Multimedia and security*, pages 164–174, Geneva, Switzerland, 2006. ACM.

[110] Sha Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17:281–308, 1988.

Vita

Gaurav Gupta, son of Ishwar Chandra and Anupama Gupta, was born in Ratlam, India at 04:55 on 25th November, 1980 and went to St. Paul H.S.S., Indore for primary and secondary education.

He finished his undergraduate degree from Devi Ahilya University, Indore, Indiaand postgraduate degree from National University of Singapore, Singapore.

His research interests include information security, intellectual property, copyright protection, information hiding, watermarking, and fingerprinting. He has also lectured several undegraduate and postgraduate topics while at IIPS, India and Macquarie University, Australia. His teaching interests include programming languages including C++ and Java, object oriented technology, data structures, and information systems.

Permanent Address: 40, Aditya Nagar
A B Road
Indore - 452017
INDIA

This dissertation was typeset with LaTeX 2_ε[1] by the author.

[1] LaTeX 2_ε is an extension of LaTeX. LaTeX is a collection of macros for TeX. TeX is a trademark of the American Mathematical Society. The macros used in formatting this dissertation were written by Dinesh Das, Department of Computer Sciences, The University of Texas at Austin, and extended

by Bert Kay, James A. Bednar, and Ayman El-Khashab.

Index

attribute, xxiii, xxiv, 23, 25, 26, 59, 62, 63, 65, 66, 68–70, 110, 113–118, 120, 132, 133, 136

blind, 56, 123–126, 138, 141
branch, 44, 46, 47, 91, 93, 94, 98, 99, 103–108, 140

copyright, 1, 33, 142

detection, 36, 50, 51, 61, 65–67, 69–72, 109–113, 115, 116, 120, 123, 125, 127, 130, 131, 134–137, 140

embedding, 36, 46, 48, 50, 56, 61, 65, 79, 81, 84, 86, 96, 104
extraction, 48, 49, 51, 77

FBF, 91, 94, 95, 98, 100, 103–106, 108

hash, 15, 16, 65, 75, 95–97, 99, 105, 106

ICBF, 94, 96, 105, 106, 108
imperceptibility, 2, 140
insertion, 39, 53, 65–67, 69, 70, 77, 83, 95, 109, 110, 116, 123, 130, 137, 140
invisible, 41

permutation, 46, 77–79, 81, 92
primary key, xxiii, 25, 62, 63, 65–67, 69, 70, 110, 111, 113, 124, 129, 130

relation, xxiii, 51, 56, 57, 61, 63, 64, 66–72, 109–121, 123–126, 129, 130, 132, 133, 136, 137, 140, 141
reversible, 109, 110, 113, 115, 116, 120, 123–126, 129, 130, 138, 141, 142
robustness, 1, 72, 139, 140, 142

secondary watermarking, 70, 73, 109, 113, 115, 124, 129, 137, 140–142
secret key, 141
semi-blind, 70

transformation, 75