Doctoral Thesis

A Study of Digital Watermarking for Protecting the Multimedia Content

September, 2023

Tatsuya Yasui

Graduate School of
Natural Science and Technology
(Doctor's Course)
OKAYAMA UNIVERSITY

博士論文

マルチメディアコンテンツを保護するための電子透かしに関する研究

2023年9月

安井達哉

岡山大学大学院自然科学研究科

Doctoral Thesis

A Study of Digital Watermarking for Protecting the Multimedia Content

Author:
Tatsuya Yasui

Supervisor:
Minoru Kuribayashi

Co-supervisors:
Nobuo Funabiki
Yasuyuki Nogami

A dissertation submitted to

Okayama University

in fulfillment of the requirements for the degree of

Doctor of Philosophy in Engineering

in the

Graduate School of Natural Science and Technology

Acknowledgement

First of all, I would like to show my greatest appreciation for Associate Professor Minoru Kuribayashi, the supervisor throughout my Bachelor's, Master's, and Doctor's courses at Okayama University. I am greatly indebted to him, whose encouragements, advices, and supports from the beginning enabled me to develop the understanding of this subject, not only in scientific but also in life. He gave me valuable advices, comments, and guidances when writing papers and presenting them. Thanks for the kind support.

As well as Prof. M. Kuribayashi, I am also grateful for Professor Nobuo Funabiki who is my co-supervisors gave many knowledge and support both in research activities and student life in the Distributed System Design Laboratory. He also kindly supported me in my job hunting activities. Thanks to him, I was able to work and do research at the same time.

Professor Yasuyuki Nogami is my co-supervisors gave me a lot of opportunities to enhance my skill as a researcher. He was the first professor I met at Okayama University when I was a high school student. He gave me this great opportunity to study here. At the same time, he taught me a lot of knowledge about security and mathematics in my Bachelor's, Master's, and Doctor's courses. Those have been sufficiently exploited in my research activities and will be fundamental pieces of knowledge throughout my future activities.

Reviewing my student life in the Distributed System Design Laboratory, colleagues who gathered around me during my Bachelor's, Master's, and Doctor's courses encouraged my activities and brought a lot of the memories of my student life. Many thanks are for them and nothing would exceed them. I hope they will succeed in their own ways with many wishes.

Ms. Keiko Kawabata always helped me with kind deals in the processes of many types of official documents. It was nice and honor of me that I could share the time with her.

I would like to acknowledge Dr. Teddy Furon of INRIA for his help in computing the WCA parameters to the Nuida code.

It is my great pleasure to thank all those who have supported and encouraged me throughout my study for Doctor's degree. It would not have been possible to complete this work without your kind help. You are all great people, full of blessings, and may you

continue to inspire and support others.

Finally, I would like to show my great thanks to my family for allowing me to learn in the doctorial course. They have always cared about my health and hoped to be a success in this field with many fortunes. I would like to conclude this acknowledgment with many thanks for my family and surroundings.

Research and social activities

Refereed journal papers

1. **T. Yasui**, M. Kuribayashi, N. Funabiki, and I. Echizen, "Near-Optimal Detector for Binary Tardos Code by Estimating Collusion Strategy," IEEE Trans. Information Forensics and Security, vol. 15, pp. 2069-2080, 2020. DOI: 10.1109/TIFS.2019.2956587

2. **T. Yasui**, T. Tanaka, A. Malik, M. Kuribayashi, "Coded DNN Watermark: Robustness against Pruning Models Using Constant Weight Code," Journal of Imaging, vol.8, no.6, pp. 152-167, 2022. DOI:10.3390/jimaging8060152

3. M. Kuribayashi, **T. Yasui**, A. Malik, "White Box Watermarking for Convolution Layers in Fine-Tuning Model Using the Constant Weight Code," Journal of Imaging, vol.9, no.6, pp. 117-135, 2023. DOI:10.3390/jimaging9060117

International conference proceedings (with review)

1. **T. Yasui**, M. Kuribayashi, N. Funabiki, and I. Echizen, "Estimation of Collusion Attack in Bias-Based Binary Fingerprinting Code," Asia-Pacific Signal and Information Processing Association Annual Summit and Conf. (APSIPA ASC 2018), pp.1550-1555, 2018.

2. M. Kuribayashi, T. Tanaka, S. Suzuki, **T. Yasui**, and N. Funabiki, "White-Box Watermarking Scheme for Fully-Connected Layers in Fine-Tuning Model," 9th ACM Workshop on Information Hiding and Multimedia Security(IH&MMSec'21), pp.165-170, 2021.

Domestic conference proceedings (without review)

1. **安井達哉**, 栗林稔, 舩曳信生, "電子指紋符号における結託攻撃の戦略推定," 信学技報, vol. 117, no. 476, EMM2017-80, pp. 17-22, 2018 年 3 月.

2. **安井達哉**, 栗林稔, 舩曳信生, 越前功, “電子指紋符号における不正者検出のための動的戦略推定,” 信学技報, vol. 118, no. 224, EMM2018-58, pp. 65-70, 2018 年 9 月.

3. **安井達哉**, 栗林稔, 舩曳信生, “雑音環境における電子指紋符号に対する結託攻撃の攻撃戦略推定,” 信学技報, vol. 118, no. 494, EMM2018-107, pp. 83-88, 2019 年 3 月.

4. **安井達哉**, 栗林稔, 舩曳信生, “電子指紋符号の結託攻撃パラメータ推定のための特徴ベクトル導出及びその次元削減,” コンピュータセキュリティシンポジウム 2019 論文集, vol. 2019, pp. 982-989, 2019 年 10 月.

5. 田中拓朗, **安井達哉**, 栗林稔, 舩曳信生, “DM-QIM によるディープニューラルネットワークの重みに対する電子透かし法,” 信学技報, vol. 119, no. 463, EMM2019-106, pp. 25-30, 2020 年 3 月.

6. 田中拓朗, **安井達哉**, 栗林稔, 舩曳信生, “埋め込みロス関数なしの DNN 電子透かしの収束に関する考察,” 信学技報, vol. 121, no. 247, EMM2021-62, pp. 49-54, 2021 年 11 月.

7. **安井達哉**, Malik Asad, 栗林 稔, “重み一定符号を用いた DNN 電子透かしの検出法,” 情報科学技術フォーラム講演論文集 (FIT), vol. 21, no. 3, pp. 145-150, 2022 年 8 月.

Abstract

With the spread of the Internet, digital contents have become more accessible and have greater impacts on social activities. Digital contents such as images, audio, video, and texts are collectively called multimedia content and have permeated our lives in various forms. However, the growing influence of multimedia content has raised the necessity to find value in the content itself and protect it as intellectual property. Since multimedia content can be easily duplicated due to its nature, a system to prevent unauthorized duplication is necessary when multimedia content is distributed for sales or other purposes. Based on this background, the field of multimedia security has been developed as a technology to protect multimedia content. Among these, the technique of suppressing reproduction and manipulation by embedding information in areas of the content that are imperceptible to humans is called digital watermarking and is the subject of much research.

Watermarking algorithms must be designed for the actual environment in which they will be used. For example, when embedding a watermark in an image, the algorithm must be robust enough to correctly extract the watermark even when the image is degraded by lossy JPEG compression. The image degradation caused by embedding the watermark must be minimized, and the amount of information in the watermark that can be embedded must be as large as possible. Thus, watermarking requires consideration of *Robustness*, *Fidelity*, and *Capacity*, which are generally known to have a trade-off relationship. In particular, it is necessary to carefully define fidelity in accordance with diversified multimedia content. In this study, we propose two different approaches to watermarking.

Recently, research and development of deep learning techniques have been actively conducted due to the improvement of computer performance and the ability to handle big data. Deep Neural Network (DNN) models are extremely costly because they require expensive computational resources and large amounts of training data. The fidelity that a DNN watermark must satisfy is the accuracy of the learned model. This means that embedding the watermark should not degrade the accuracy of the learned model for the task. For robustness, it is natural to embed watermarks in redundant parts of the model, such as the weight parameters, because once a learning model is generated, it is unlikely to be distorted by operational assumptions. However, depending on the size and con-

straints of the system, it may be desirable to introduce a lighter sized learning model while maintaining the accuracy of the original model. In such cases, model compression is a common technique. Among these methods, pruning is a technique to reduce the size of the model by removing unnecessary weights with low contribution for the accuracy. Pruning can effectively reduce the size of the training model while maintaining accuracy. A pruning attack is envisioned to take advantage of this property to intentionally remove watermarks embedded within the weight parameters. We propose a DNN watermarking that is robust against pruning attacks, which are assumed to remove watermarks embedded by DNN watermarking as the first approach. It is achieved by assigning the symbol "0" of the constant weight code to the weights that is most likely to be deleted by the pruning attack.

One of the threats that focus on watermarks embedded in multimedia content is the collusion attack. The collusion attack is the attack in which multiple colluders attempt to remove watermarks from content. As a countermeasure against collusion attacks, research has been conducted on codes called fingerprint codes. Fingerprint codes include Tardos codes and Nuida codes, which can generate code words of the smallest order of code length. Furthermore, optimal detectors that can theoretically detect the maximum number of colluders for these codes have been proposed. However, an optimal detector is difficult to achieve because it requires two parameters that are unknown to the detector: the number of colluders and the collusion strategy. We propose an estimator in which two types of parameters are estimated from pirated codewords generated by colluders for practical use of the optimal detector as the second approach. This estimator also takes the real environment affected by noise into account.

The proposal of a DNN watermarking representative of diversified multimedia content and the realization of an optimal detector for digital fingerprinting codes to be embedded in the watermark are expected to form a comprehensive system to protect the multimedia content.

Future works on DNN watermarking includes the study of DNN watermarks that are robust to attacks other than the pruning attack targeted in this study, and the study of methods with weight parameters that are distributed in a way other than a uniform distribution. Future works on fingerprinting codes includes a study on collusion attacks under the assumption that an optimal detector is available.

概要

インターネットの普及によってデジタルコンテンツが身近になり社会活動に与える影響も大きくなってきた．画像，音声，動画，テキストなどのデジタルコンテンツは，総称してマルチメディアコンテンツと呼ばれており様々な形で生活に浸透している．一方で，マルチメディアコンテンツの影響度が大きくなることにより，コンテンツそのものに価値を見出し知的財産として保護する必要性が生じてきた．マルチメディアコンテンツはその利用のしやすさから簡単に複製できてしまうため，マルチメディアコンテンツを販売などで配布する際には，不正に複製されないような仕組みが必要である．このような背景に基づき，マルチメディアコンテンツを保護するための技術としてマルチメディアセキュリティという分野が発達してきた．その中でも，人間の知覚の特性を利用して人間が知覚できないコンテンツの領域に情報を埋め込むことで複製や改ざんを抑制する仕組みは電子透かしと呼ばれており，盛んに研究が行われている．

電子透かしは実際に使われる環境を考慮してアルゴリズムを構成する必要がある．例えば，画像に対して透かしを埋め込む場合には，非可逆圧縮である JPEG 圧縮によって画像が劣化した場合に透かしを正しく抽出できるための堅牢性を備えていなければならない．また，透かしを埋め込んだことによる画像の劣化は最小限に抑えなければならない．同時に，埋め込むことができる透かしの情報量は可能な限り大きくある必要がある．このように，電子透かしにおいては堅牢性 (Robustness), 忠実性 (Fredity), 容量 (Capacity) を考慮する必要があるが，一般的にこれらはトレードオフの関係にあることが知られている．特に忠実性においては，多様化するマルチメディアコンテンツにあわせた定義に注意しなければならない．本研究では，電子透かしに関する研究として異なるアプローチで 2 つの方式を提案する．

昨今，計算機の性能向上やビッグデータを取り扱うことができるようになったことで，深層学習技術の研究開発が盛んに行われている．深層学習によって生成されたモデル (DNN モデル) は，高価な計算機リソースと大量の学習データを必要とするため，大きな価値がある．そのため，他のマルチメディアコンテンツと同様にその価値を保護する技術が必要である．DNN モデルに対する電子透かしは DNN 電子透かしと呼ばれており満たすべき忠実性は，学習モデルの精度である．つまり，電子透かしを埋め込むことによって学習モデルのタスクに対する精度が低下してはならない．堅牢性に関しては，一度生成した学習モデルが運用の仮定で歪むことは少ないため，学習モデルの重みパラメータ等の冗長な箇所に電子透かしを埋め込むことが自然な発想である．しかし，システムの規模や制

約によっては，既存の学習モデルの精度をそのままに，より軽量な学習モデルをデプロイしたい場合がある．このような場合に一般的に行われる技術として，モデル圧縮がある．その中でも，寄与が低い不要な重みを除くことでモデルのサイズを軽量化する手法としてプルーニングがある．プルーニングを行うことで学習モデルの精度はそのままにサイズを軽量化することができるが，電子透かしを埋め込んでいた場合には，元の学習モデルの重みパラメータを変更するため，堅牢性が低下してしまう．また，この性質を利用して意図的に電子透かしを除去するプルーニング攻撃を想定することができる．そこで本研究のアプローチでは，DNN 電子透かしによって埋め込まれた電子透かしを取り除くために想定されるプルーニング攻撃に対して堅牢な電子透かしを提案する．具体的には，プルーニングによって取り除かれる可能性のある箇所に重み一定符号のシンボル “0” を配置することでプルーニングによる影響を減らして実現する．

マルチメディアコンテンツに埋め込む電子透かしに着目した場合の脅威の一つに結託攻撃がある．結託攻撃とは，複数の結託者によってコンテンツの電子透かしを取り除こうとする攻撃である．結託攻撃に対しては電子指紋符号と呼ばれる符号の研究が行われており，その中でも最小オーダの符号長で符号語を生成できる Tardos 符号や Nuida 符号が提案されている．さらに，これらの符号語に対して理論上最も多くの結託者を検出できる最適な検出器も提案されている．しかし，最適な検出器は，検出者が知りえないパラメータである結託者数と結託攻撃戦略の 2 種類のパラメータを必要とするため実用が困難であった．そこで本研究のアプローチでは，結託者によって生成された不正符号語から 2 種類のパラメータを推定する推定器を提案することで最適な検出器の実用を目指す．さらに，実環境を想定して不正符号語がノイズによって歪んでいる場合でも推定できる推定器も提案する．

本研究で提案した，多様化するマルチメディアコンテンツの 1 つである DNN モデルを対象とした DNN 電子透かしアルゴリズムの提案と，電子透かしに埋め込む電子指紋符号の最適な検出器の実現によりコンテンツを保護するための包括的なシステムを形成することが期待できる．

DNN 電子透かしの今後の課題では，本研究で対象としたプルーニング攻撃以外の攻撃に堅牢な DNN 電子透かしの検討や，一様分布以外の分布をなす重みパラメータにおける手法の検討が考えられる．電子指紋符号の今後の課題は，最適な検出器がある前提で想定される結託攻撃に対する検討が考えられる．

Contents

Chapter 1

Introduction

This chapter begins by introducing the purpose of protecting the copyright of multimedia content is presented with reference to multimedia security techniques and related research.

1.1 Copyright Protection for Digital Content

Digital content has diversified due to the dizzying pace of technological innovation and changes in the environment that surrounds us. It includes images, sounds, audios, videos, and texts, and is referred to as multimedia content. They are essential to form our society, and at the same time, valuable multimedia content must be protected. Their accessibility makes copyright problems such as piracy through copying [8]. One approach to solving this problem is encryption. Content is encrypted and only the user with the private key can decrypt and view the content. This is similar to whitelist-style access restrictions. The disadvantages of content encryption include usability reduction and the fact that the original content cannot be obtained if decryption by an authorized user fails. Another approach is digital watermarking. In contrast to encryption, digital watermarking protects content without reducing its usability. However, instead of usability, watermarking does not keep the original content secret and tolerates copying. Common algorithms protect content by embedding information in verbose parts of the original content and achieving traceability with the presence of that signal.

1.2 Digital Watermarking and Taxonomy

Digital watermarking is a technique for secretly embedding information in verbose parts of digital content. Digital content must not be distorted by watermarks and must be able to extract information accurately. This technique allows for the confidential exchange of messages and the protection of the intellectual property of digital content by tracking

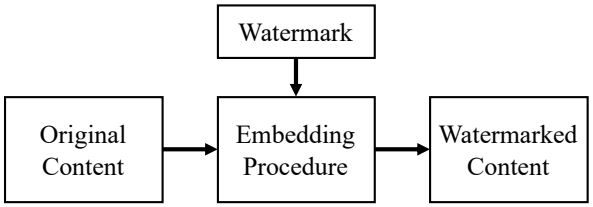


Figure 1.1: The watermarking embedding

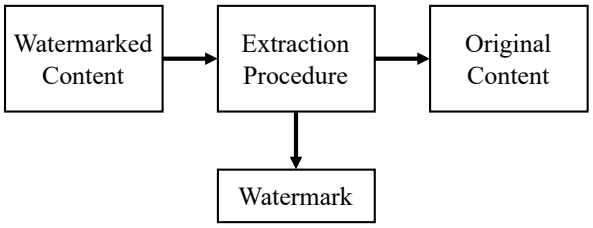


Figure 1.2: The watermarking extraction

embedded information. Figure 1.1 below schematically illustrates the embedding of a watermark using the watermarking procedure. Extraction is also shown in fig. 1.2.

Digital watermarking has three Trade-offs (*Fidelity*, *Capacity*, *Robustness*)(fig.1.3). Fidelity is the minimal impact of the watermark on the original content, and it should be unnoticeable when watermark is invisible. Robustness is the ability to decrypt the watermark even if some changes are made to the content in which the watermark is embedded. There are so many causes by which watermark is degraded, modified during transmission, or attacked. Therefore, watermark should be robust enough to withstand any attack or threat. And capacity is the size of the payload that can be propagated by

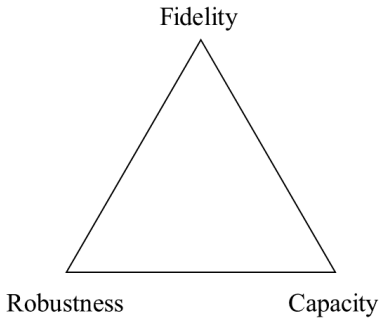


Figure 1.3: The watermarking trade-off triangle

the watermark.

Digital watermarking can be classified into two categories according to their purpose. The first is a fragile watermarking that guarantees the originality of the host content. It can detect content manipulation if the watermark embedded in the host content is not detected correctly. The most likely manipulation situation for images is image processing. In this situation, it is most effective to embed the watermark at the point where it is changed by the image processing. The other purpose is to protect the content using reversibility by providing robustness to the embedded watermark. Watermarks embedded in host content are corrupted in many situations. Using an error-correcting code for watermarking makes the watermark robust against manipulation and editing and is called robust watermarking.

Digital watermarking for multimedia content has been studied immensely over the past two decades [3,14,56,57]. In the case of images, the least significant bit (LSB) of the pixel value is substituted and embedded in the spatial domain [13,52,62]. Discrete Cosine (DCT) [43], Discrete Wavelet Transform (DWT) [69], and Fourier Transform (DFT) [2,67] embed watermarks in the frequency domain and are widely used for imperceptible and robust watermarking.

Digital watermarking can be applied to content that has embeddable areas, but in recent years, software such as DNN models has also become part of multimedia content. This study focuses on digital watermarking for traditional multimedia content like images and music, as well as on the study of digital watermarking for software like DNN models that has gained attention in recent years.

1.3 DNN Watermarking

Digital watermarking has been studied for a long time to preserve the copyright of digital data such as image, audio, and video by inserting some confidential information. In addition, the widespread use of DNN models in the current scenario is crucial to protect their copyrights. Researchers have been studying DNN watermarking to protect the intellectual property associated with DNN models. Because of the multiple network layers in a DNN model, many parameters known as network weights must be trained to attain a local minimum. However, several degrees of freedom are available for choosing the weight parameters for embedding a watermark. Moreover, the watermark is inserted in such a way that the accuracy of the model on its original task decreases to the lowest extent possible.

DNN watermarking techniques can be categorized into two types [10]: white-box watermarking, black-box watermarking. In white-box watermarking, internal architecture and parameters are exposed to the public, and black-box watermarking takes advantage

of the functionality of DNN models. In some cases, when a specific query is input, the watermark can be retrieved from its output without knowing the internal parameters; this characteristic is equivalent to creating a backdoor into the model. Basically, in the black-box methodology can only access the final layer's output, some experts have investigated training networks to intentionally make wrong output for a given input and then use it as a watermark [40, 74]. Moreover, the research in black-box watermarking specially in frequency domain [29, 73] is also taking a great attention, which performs well in terms of imperceptibility and robustness.

The first white-box method was presented in [50, 68], where a watermark was embedded into the weight parameters of a convolutional neural network (CNN) model. The embedding operation is performed simultaneously along with the training of the model by introducing an embedding loss function so that the weights are updated according to the watermark and the supervised dataset.

In [11, 58], the selection of the feature vector in the methods presented in [50, 68] was refined. The paper [12, 15] reported that almost all local minima are very similar to the global optimum. Empirical evidence has shown that a local minimum for deeper or larger models is sufficiently good because their loss values are similar. With this characteristic, the watermark was embedded into some sampled weight values in [34, 71]. In [71], the sample weight values were inputted to a DNN model that is independent of the host model, and error back-propagation was used to embed the watermark in both the host model and the independent model.

The white-box method must be sufficiently robust to recover the watermark from a perturbed version of a DNN model because attackers can directly modify the parameters. One instance of perturbation is model pruning, where redundant neurons are pruned without compromising accuracy to reduce the computational costs of executing DNN models. The purpose of pruning is to remove less-important weight parameters from the DNN model whose contribution to the loss is small. If the watermark signal is embedded into such less-important parameters, it is easily removed or modified by pruning. Therefore, a crucial requirement of DNN watermarking is the robustness against pruning attacks [45] while ensuring that the watermarked parameters are relevant to the original task. [68] showed experimentally that the watermark does not disappear even after a pruning attack that prunes 65 % of the parameters. Another study achieved robustness against 60 % pruning [41]. This study adopted the idea of spread transform dither modulation (ST-DM) watermarking by extending the conventional spread spectrum (SS)-based DNN watermarking. A detailed survey of DNN watermarking techniques can be summarized in [42].

Another study in [77], embedding the watermark into the model structure by pruning has been proposed. This method was shown to be robust against attacks that adjust the model's weights, which is a threat in other embedding methods. Moreover, this method

considers pruning an embedding method, whereas we consider pruning as a perturbation by the attacker and propose a robust embedding method against pruning. In communication channels, pruning can be regarded as an erasure channel between the transmitter and the receiver of the watermark. Because numerous symbols are erased over the channel (e.g., more than half of the weight parameters are erased by pruning), erasure correcting codes are unsuitable for this channel.

1.4 Digital Fingerprinting

Digital fingerprinting assigns a unique ID to each content to ensure that it is authentic with a unique ID. If multiple managed IDs are detected, the IDs can be used to track down unauthorized persons, thus providing traceability. This is similar to the management by product keys or license keys. When applied to multimedia content, IDs cannot be printed, so they are often embedded directly into the content using a digital watermark. In this case, robust watermarking is suitable because the watermark must be reversible.

Collusion-secure codes have been studied against collusion attacks. In the field of collusion-secure codes [5, 44, 64, 66, 76], Tardos code [65] is known to produce bias-based fingerprinting in which each symbol of a colluder's codeword is determined by a certain biased probabilistic distribution. As the code length is theoretically of the minimum order, the performance of a Tardos code has been intensively investigated to improve its traceability. In particular, Nuida et al. [53, 54] constructed an interesting variant using a discrete probabilistic (Gauss–Legendre) distribution to customize the bias-based fingerprinting code for a fixed number of possible colluders. For convenience, their fingerprinting code is referred to as the Nuida code in this paper.

To identify illegal users (colluders) from the pirated codeword, a tracing algorithm (detector) is used to find suspicious users by calculating the similarity among the colluder's codewords. Existing detectors can be classified into three types: catch-one, catch-many, and catch-all [75]. With the catch-one technique, the most suspicious user is the one with the maximum similarity score and is assumed to be guilty. The assumption here is that there is a collusion among several illegal users, so a catch-many type detector is desirable because it can identify as many illegal users as possible. Although all colluders can be identified using a catch-all approach, the false-negative rate (i.e., no colluders are detected) is higher. Therefore, we focus here on catch-many detectors.

A good tracing algorithm can catch as many colluders as possible with a constant small false-positive rate. The tracing algorithm is essentially composed of two operations: scoring, which calculates similarity scores, and classification using a threshold. The colluders can choose an arbitrary collusion strategy, such as majority and minority voting, to generate a pirated codeword. As Tardos and its revised scoring functions [70] are independent

of the collusion strategy, the functions cannot achieve the high performance. Furon and Perez-Freire [21] proposed an optimal detector based on information theoretical analysis that calculates the highest score using information about the collusion strategy and the number of colluders. Because of the difficulty of estimating these parameters [21], several researcher groups [1, 18, 39, 47, 55] have investigated defense strategies to minimize the performance gap from this optimal detector. For instance, scoring functions that adjust their weighting parameters based on each symbol have been developed [39, 55]. These scoring functions require no information about the collusion strategy because they use only the symbol combination of the colluders' codewords and the pirated codeword.

In this thesis, we have developed an effective estimator for these parameters that uses the characteristics of the discretized probabilistic distribution of the Nuida code. This estimator has two steps.

1.5 Outline and Contributions

This thesis is organized with five chapters as follows.

In Chapter 2, the watermarking and fingerprinting techniques and algorithm are reviewed.

Chapter 3 proposes the DNN watermarking robust against pruning attacks. Our contribution is the introduction of encoding technique into the DNN watermark to make it robust against pruning attacks. While previous studies have proposed DNN watermarks that are robust against a certain level of pruning rate, our method can assure the robustness with a pre-defined level of pruning rate by carefully setting the encoding parameters. The common scenario in which DNN watermarks are used in DNN model is the buying and selling of DNN models. In this scenario, our method can prevent illegal redistribution and illegal copying by users who have purchased the DNN model. As a white-box watermarking is assumed in our method, it suffers from the direct modification of weight parameters, which is the common threat in white-box setting. If the weight parameters are replaced with random values and trained from the scratch with enough dataset, the watermark can be removed completely without compromising the performance of DNN model. Hence, it is assumed in our method that the attacker cannot train the target DNN model from scratch in terms of computational resources and amount of training dataset.

Chapter 4 proposes the near-optimal detection for near-optimal detection for binary Tardos code by estimating collusion strategy. Our contribution is the realization of detector for optimal detector by estimating requirement parameters which are number of colluders and collusion strategy by colluders. In other words, we have developed an estimator that estimates the parameters required for an optimal detector. The collusion strategy estimator focuses on the number of symbols in the binary code. Symbols are

distributed differently depending on the number of colluders and collusion strategies. It achieved better performance than any of the previous studies. Similar performance was also achieved when the evaluation was conducted under realistic noise-added scenarios.

Finally, Chapter 5 concludes the thesis by briefly reviewing the entirety.

Chapter 2

Preliminaries

This chapter describes the techniques related to digital watermarking and digital fingerprinting.

2.1 Notation

The notation of parameters used in this paper are summarized below:

- a: The regular italic style represents a scalar such as an vector elements or a constant.
- $\boldsymbol{b}$: The bold italic style represents a vector. such as an array, a signal or a codeword.
- $sort()$: Sort algorithm with ascending order.
- $sgn()$: Sign function.
- $act()$: Activation function like a sigmoid function.

2.2 DNN Watermark

Many watermarking techniques have been devised to protect multimedia content such as audio, still images, video, and text. A watermark signal is inserted into a host signal selected from the multimedia content using a secret key. This technique can be extended to DNN models. During the training phase, the weight parameters are optimized to minimize the loss function, which represents the difference between the predicted class label and the true label. Since DNN models have a large number of weight parameters, there are many degrees of freedom for parameter tuning during the training phase. This degree of freedom allows watermarks to be inserted without compromising the performance of the DNN model. Figure2.1 shows embedding watermark in the weight parameters extracted from

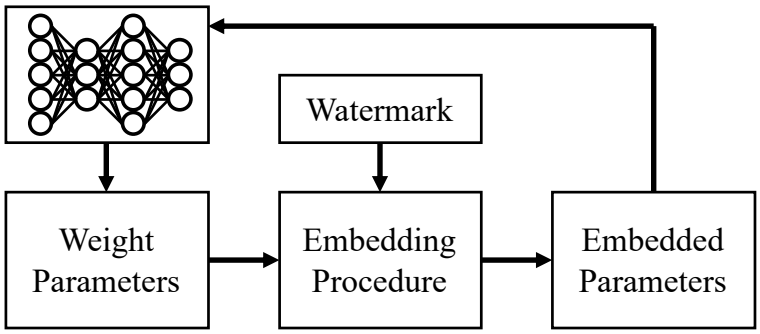


Figure 2.1: The DNN Watermarking

the DNN model. Watermarking techniques need to control the trade-off requirements of *Capacity*, *Robustness*, and *Fidelity* [42]. The fidelity that a DNN watermark must satisfy is the accuracy of the learned model. This means that embedding the watermark should not degrade the accuracy of the learned model for the task.

Generally, the weight parameters of the DNN model are initialized before training and refined to reach a single local minimum after a series of epochs. As the parameters are increased, the DNN model has many local minima with similar performance as the global minimum [12, 15]. Thus, while the process of finding a local minimum from randomly set initial values is the usual process of learning a DNN model, learning a DNN model with embedded watermarks to weights is equivalent to changing the initial values and selecting different local minima.

2.2.1 Fine-Tuning

In general, training DNN models is very expensive because of the computational resources and large training data sets used. Therefore, already trained DNN models are often used for development. To adapt a model that has been pretrained for one task to a new task, the pretrained layers are frozen and replaced with new Fully-Connected (FC) layers. For the trainable layers above the frozen layer, a new DNN model was trained on the new dataset. If a watermark was embedded in the FC layer, the watermark was completely removed by creating a fine-tuned model, which replaced the weight parameters of the watermark and changed the weight parameters of the unfrozen layer. Therefore, the watermark must be robust against the retraining unfrozen layers during fine-tuning. Figure 2.2 shows the watermarks embedded in the unfrozen layer are corrupted by retraining the weights through fine tuning.

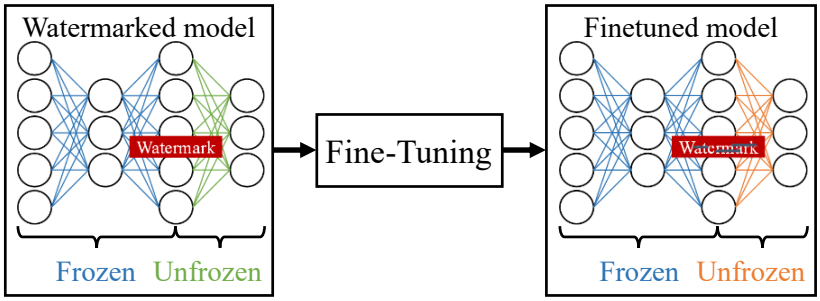


Figure 2.2: The watermarks corrupted by fine-tuning

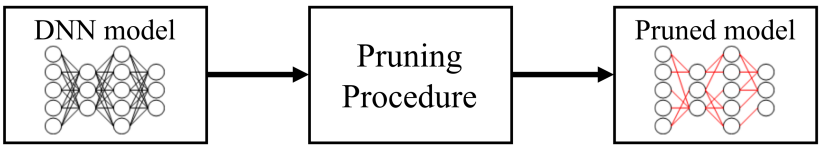


Figure 2.3: The pruning procedure

2.2.2 Pruning Attack

Owing to many neurons in DNN models, there is significant redundancy in the network; making a network deeper is a promising way to improve the performance. It is reasonable to prune such redundant neurons to reduce the model size as well as computational costs [17]. Figure 2.3 shows the cutting the paths of the DNN model by pruning. It is an NP-hard problem to find the best combination of neurons to be pruned, from among the millions of parameters in a DNN model [25]. Some heuristic pruning methods have been developed to identify relatively less important components in DNN models and retrain the pruned model to recover the model's performance. Thus, to create a robust DNN watermark, it is necessary to consider the effects of pruning as well as the changes during retraining.

The pruning methods can be roughly classified into three categories. One is weight-level pruning, which sets less important weights to zero and does not change the network structure. The other two are channel-level and layer-level pruning, which can change the network structure but require large computations to find an efficient network modification with little compromise in performance. Therefore, we focus on weight-level pruning in this study. In the weight-level pruning, after training, the parameters whose absolute values are smaller than a threshold are cut-off to zero to compress the DNN model. The threshold is set such that the model's accuracy does not decrease significantly.

For a given rate $0 \leq R < 1$, the pruning attack changes the weight values $w_i = 0$ if

$|w_i| < \tilde{w}_p$ for $0 \leq i \leq N-1$, where

$$\tilde{\boldsymbol{w}} = sort(|\boldsymbol{w}|) = sort(|w_0|, |w_1|, \ldots, |w_{N-1}|), \tag{2.1}$$

$sort()$ is a sort algorithm, and

$$p = \lfloor RN \rfloor. \tag{2.2}$$

Thus, according to the rate R, the weight parameters whose absolute values are smaller than the p-th weight are pruned.

Han et al. [26] proposed to prune network weights with small magnitude by incorporating a deep compression pipeline to create efficient models. Some criteria and settings have been proposed for weight-level pruning [19, 48]. Some types of weight-level pruning can be viewed as a process to find a mask to determine which weights to preserve.

2.2.3 Embedding Loss

DNN watermarking can be categorized into two types [10]. One is a white-box watermarking method, in which the internal details are exposed to the public. The other is a black-box watermarking method where the owner of the model only has API access to the remotely deployed model. The first white-box method was developed by Uchida et al. [50, 68]. Rouhani et al. [58] presented an improved version of it, and their research group proposed an application of fingerprinting to track illegal users. For a given DNN model, a bit string of watermark information is embedded in one or more network layer parameters. Considering the performance degradation of the model (*Fidelity*), the conventional methods described above avoid directly modifying the parameters for embedding the watermark. Therefore, they introduced binary cross entropy in the cost function during training and regularized the watermark embedding task so that the performance of the original DNN model would not be compromised. Watermark information is denoted by a vector w of length k. Let $X \in \mathbb{R}^{k \times n}$ be a matrix to be kept secret, and $\boldsymbol{p}$ be vector of weights in network layers to be watermarked which length is n. Then, the binary cross entropy $H(p)$ is defined by

$$H(\boldsymbol{p}) = -\sum_{i-1}^{k} (w_i \log(y_i) + (1 - w_i) \log(1 - y_i)), \tag{2.3}$$

where

$$y_i = \mathrm{Act}(\sum_{j=0}^{n} X_{ij} p_j) \tag{2.4}$$

Each watermark bit w_i is embedded so that the following equation becomes true.

$$w_i = \begin{cases} 1 & y_i \leq 0.5 \\ 0 & y_i < 0.5 \end{cases} \tag{2.5}$$

The cost function is composed of two functions as follows:

$$E(\boldsymbol{p}) = E_0((\boldsymbol{p})) + \lambda H(\boldsymbol{p}), \tag{2.6}$$

where $E_0(\boldsymbol{p})$ is the original cost function, and $H(\boldsymbol{p})$ is a regularization term that imposes a certain restriction on parameters $\boldsymbol{p}$, and λ is an adjustable parameter. The embedding matrix X is considered as a secret key and must be generated carefully because the distribution of the weight parameters of the watermark becomes unnatural. Wang et al. [72] pointed out the problem of significant differences in the distribution from the watermarked parameters and presented a method to remove the watermark by an overwriting attack. The main reason of this problem is that the above method changes the weights significantly in order to satisfy the condition given by Eq.2.5.

2.3 Digital Fingerprinting

In this study, the fingerprinting code is composed of N codewords with L symbols. Let $x_{j,i} \in \{0, 1\}(1 \leq i \leq L)$ represent the j-th user's codeword.

2.3.1 Collusion Attack

Suppose that c colluders attempt to produce a pirated copy from their fingerprinting codes. Under the marking assumption [5], a pirated codeword $\boldsymbol{y} = (y_1, y_2, \ldots, y_L)$ is constructed using a collusion strategy. A group of colluders is denoted by $\boldsymbol{C} = \{j_1, j_2, \ldots, j_c\}$. The collusion attack is the process of taking sequences in $\boldsymbol{I_i} = \{x_{j_1,i}, x_{j_2,i}, \ldots, x_{j_c,i}\}$ as inputs and the pirated sequence $\boldsymbol{y}$ as an output. When a pirated codeword is produced from the colluders' codewords, the marking assumption [5] states that the colluders have $y_i \in \boldsymbol{I_i}$. They cannot change the bit in the position where all of the indexes in $\boldsymbol{I_i}$ are identical because their positions are undetectable.

Furon et al. defined a collusion attack as parameter vector $\boldsymbol{\theta_c^{str}} = (\theta_0^{str}, \ldots, \theta_c^{str})$ with $\theta_\lambda^{str} = \Pr[y_i = 1 | \Phi = \lambda](0 \leq \lambda \leq c)$, where $\Phi \in \{0, \ldots, c\}$ denotes the number of "1" symbols in the colluders' copies for a given index [22]. Figure2.4 illustrates how to create a pirated codeword when five colluders attack with the majority attack. Since the symbols that compose I_1, I_2, I_5, and I_6 are the same symbols, the symbols at each index of the pirated codeword satisfy the marking assumption. The symbols at the other indices then take the majority vote. Since some collusion strategies have a greater affect on traceability than others [22], the worst case attack (WCA), which minimizes the achievable rate of the code, can be defined from an information theoretical point of view. The marking assumption enforces $\theta_0^{str} = 0$ and $\theta_c^{str} = 1$ in the collusion strategies. Typical examples for $c = 6$ are shown by the following parameters. The minority strategy adopts

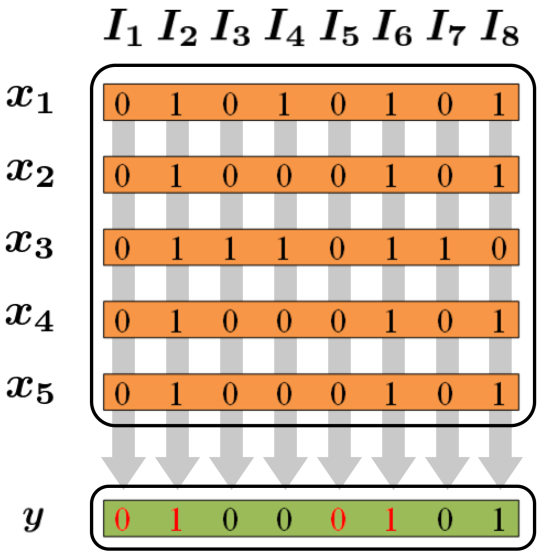


Figure 2.4: The majority attack

the symbol with the least number of symbols in the colluders' codewords. The coin-flip strategy always gives a 50% probability of "1" symbols, regardless of the symbols in the colluders' codewords. The all-0 and all-1 strategies are simple and give 0 and 1 respectively, regardless of the symbols in the colluders' codewords. The interleave strategy adjusts the probability of giving "1" symbols according to the frequency of occurrence of the symbol in the colluders' codewords. If the number of "1" symbols at a given index for 6 colluders is 2, the symbol of the pirated codeword has a $\frac{2}{6}$ probability of "1" symbols. Thus, the colluders can define any strategy to attack, but the most theoretically threatening strategy is the WCA attack.

- Majority: $\boldsymbol{\theta}_{\mathbf{6}}^{\boldsymbol{maj}}$ = (0,0,0,0.5,1,1,1)
- Minority: $\boldsymbol{\theta}_{\mathbf{6}}^{\boldsymbol{min}}$ = (0,1,1,0.5,0,0,1)
- Coin-flip: $\boldsymbol{\theta}_{\mathbf{6}}^{\boldsymbol{coin}}$ = (0,0.5,0.5,0.5,0.5,0.5,1)
- All-0: $\boldsymbol{\theta}_{\mathbf{6}}^{\boldsymbol{all0}}$ = (0,0,0,0,0,0,1)
- All-1: $\boldsymbol{\theta}_{\mathbf{6}}^{\boldsymbol{all1}}$ = (0,1,1,1,1,1,1),
- Interleave: $\boldsymbol{\theta}_{\mathbf{6}}^{\boldsymbol{int}}$ = $(0,\frac{1}{6},\frac{2}{6},\frac{3}{6},\frac{4}{6},\frac{5}{6},1)$
- WCA: $\boldsymbol{\theta}_{\mathbf{6}}^{\boldsymbol{WCA}}$ = (0, 0.5641, 0, 0.5, 1, 0.4359, 1)

For the above case, the candidates collusion strategies are denoted by str = {maj, min, coin, all0, all1, int, WCA}.

2.3.2 Fingerprinting Code

2.3.2.1 Tardos Codes

A Tardos code is a binary bias-based fingerprinting code. In the Tardos code, $x_{j,i}$ is generated from an independent and identically distributed set of random numbers with probability p_i such that $\Pr[x_{j,i} = 1] = p_i$ and $\Pr[x_{j,i} = 0] = 1 - p_i$. This probability p_i needs to satisfy the following conditions, where the maximum number c_{max} of colluders should be determined during the construction of the codeword. We select p_i in accordance with continuous $f(p)$, where $f(p)$ is given by

$$f(p) = \frac{1}{2\sin^{-1}(1-2t)} \frac{1}{\sqrt{p(1-p)}}. \tag{2.7}$$

Both the codeword $\boldsymbol{x_j} = (x_{j,1}, x_{j,2}, \ldots, x_{j,L})$ and the sequence $\boldsymbol{P} = (p_1, p_2, \ldots, p_L)$ must be kept as secret parameters.

2.3.2.2 Nuida Codes

To improve the performance of the Tardos code, Nuida et al. presented a discrete version of the bias distribution that is customized for a given maximum number of colluders c_{max} [53, 54]. Let $L_k(x) = (\frac{d}{dx})^k(x^2-1)^k/(k!2^k)$ be the k-th Legendre polynomial, and set $\tilde{L}_k(x) = L_k(2x-1)$. We define $\mathcal{P}^{GL}_{2k-1} = \mathcal{P}^{GL}_{2k}$ to be the finite probability distribution whose values are the k zeros of $\tilde{L}_k$, with each value p selected with probability $\eta(p(1-p))^{-3/2}\tilde{L}'_k(p)^{-2}$, where η is a normalized constant that ensures the sum of the probabilities is equal to 1. Similar to the Tardos code, the codewords of the Nuida code are generated using the bias probability sequence $\boldsymbol{P}$. Because of the discrete values, the candidate values for $p_i \in \boldsymbol{P}$ are finite, and the number of candidates is $n_g = \lceil c_{max}/2 \rceil$. Each probability p_i can be classified into ξ groups. Numerical examples are presented in Table 2.1, where P_ξ and Q_ξ, for $1 \leq \xi \leq n_g$, denote the values of the discretized probabilities and their emerging probabilities, respectively. For example, when $c_{max} = 8$ and length L of sequence $\boldsymbol{P}$ is 10000, the number of elements for which $p_i = P_2 = 0.33001$ is approximately $L \cdot Q_2 \approx 2517$ on average. As each symbol $x_{j,i}$ of the users' codewords is independently and identically selected under the constraint $\Pr[x_{j,i} = 1] = p_i$, the symbols of a codeword $\boldsymbol{x_j}$ can be separated into n_g groups on the basis of $p_i \in \boldsymbol{P}$. The illustration of generating the codeword $\boldsymbol{x_j}$ based on the bias probability sequence $\boldsymbol{P}$ is shown in Fig. 2.5. In Figure 2.5, indices of the same color indicate the same probability P. They give "1" symbols with the same probability P.

2.3.3 Tracing

A tracing algorithm (a "detector") is composed of a scoring function and a classification function. We consider error rates ϵ_{FP} and ϵ_{FN}; tracing algorithm Tr outputs suspicious

Table 2.1: Example of discrete Nuida code bias distribution.

c_{max}	P	Q	c_{max}	P	Q
1,2	0.50000	1.00000	9,10	0.04691	0.19829
3,4	0.21132	0.50000		0.23077	0.20104
	0.78868	0.50000		0.50000	0.20134
5,6	0.11270	0.33201		0.76923	0.20104
	0.50000	0.33598		0.95309	0.19829
	0.88730	0.33201	11,12	0.03377	0.16502
7,8	0.06943	0.24833		0.16940	0.16733
	0.33001	0.25167		0.38069	0.16765
	0.66999	0.25167		0.61931	0.16765
	0.93057	0.24833		0.83060	0.16733
				0.96623	0.16502

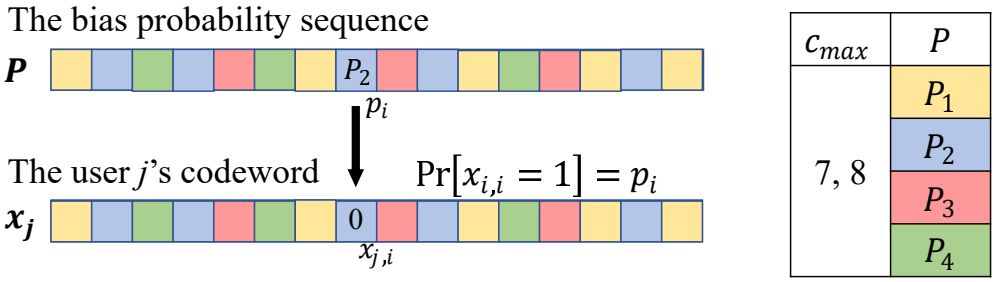


Figure 2.5: The generation of codeword

users, and $\boldsymbol{C}$ is the group of colluders.

- ϵ_{FP}: false positive
$$\epsilon_{FP} = \Pr[Tr(\boldsymbol{y}) \not\subset \boldsymbol{C} | Tr(\boldsymbol{y}) \neq \emptyset].$$
- ϵ_{FN}: false negative
$$\epsilon_{FN} = \Pr[Tr(\boldsymbol{y}) \cap \boldsymbol{C} = \emptyset].$$

Tardos proposed the following scoring function [65]:

$$S_j = \sum_{i=1}^{L} S_{j,i} = \sum_{i=1}^{L} y_i U_{j,i}, \tag{2.8}$$

where

$$U_{j,i} = \begin{cases} -\sqrt{\dfrac{p_i}{1-p_i}} & (x_{j,i} = 1) \\ \sqrt{\dfrac{1-p_i}{p_i}} & (x_{j,i} = 0) \end{cases}. \tag{2.9}$$

The tracking algorithm Tr can be classified into three categories by output.

- Catch-All: Outputs all colluders in $\boldsymbol{C}$.
- Catch-Many: Outputs as many colluders as possible in $\boldsymbol{C}$.
- Catch-One: Outputs the most suspicious colluder in $\boldsymbol{C}$.

For classification, the catch-one approach identifies the suspicious user with the maximum score as an illegal user if the score exceeds a threshold. The catch-many approach identifies all users whose scores exceed the threshold as illegal users. The scoring function in Eq. (2.8) can be used for Nuida code.

2.3.4 Threshold

In a catch-many detector, suspicious users with scores exceeding a threshold Z are regarded as illegal users. Some methods approximate the distribution of a user's score S_j by using a Gaussian distribution [31] to calculate the threshold for satisfying a given false-positive probability. Any increase in the length of the users' codewords enhances the accuracy of the approximation. However, it has been reported [60] that such an approximation is not appropriate for calculating the threshold so that the false-positive rate is less than ϵ_{FP} because the tail of the Gaussian distribution is not accurate for short codewords. For accurate measurement in the tail part, Furon et al. [22] and Cérou et al. [7] proposed an efficient method for estimating the probability of rare events. By using this rare event simulator, we can estimate ϵ_{FP} for a given threshold Z, which means that we calculate the mapping $\epsilon_{FP} = F(Z)$.

Chapter 3

Coded DNN Watermark: Robustness Against Pruning Models Using Constant Weight Code

3.1 Introduction

In this chapter, we encode the watermark using binary constant weight codes (CWC) [6,59] to make it robust against weight-level pruning attacks. The preliminary version of this paper is available at [35]. The symbols "1" used in the codeword are fixed and designed to be as small as possible. Thus, most of the symbols used in the codeword becoming are "0". To embed such a codeword, we enforce a constraint by using two thresholds while training the DNN model. The amplitude for symbol "1" is controlled to be greater than a high threshold, and that for symbol "0" is controlled to be smaller than a low threshold.Once a pruning attack is executed, the erasure of weight parameters does not affect the symbols "0" because these symbols can be extracted even if the amplitude is small. On the other hand, the symbol "1" can be detected correctly because of the high amplitude. Under the assumption that the values of weight parameters follow Gaussian or uniform distribution, the design of the two thresholds is considered to ensure robustness against pruning attacks. In the experiment, we evaluate validity of the thresholds in terms of pruning attacks and retraining of the pruned models.

Our contribution is the introduction of encoding technique into the DNN watermark to make it robust against pruning attacks. While previous studies have proposed DNN watermarks that are robust against a certain level of pruning rate, our method can assure the robustness with a pre-defined level of pruning rate by carefully setting the encoding parameters. The common scenario in which DNN watermarks are used in DNN model is the buying and selling of DNN models. In this scenario, our method can prevent illegal redistribution and illegal copying by users who have purchased the DNN model. As a

white-box watermarking is assumed in our method, it suffers from the direct modification of weight parameters, which is the common threat in white-box setting. If the weight parameters are replaced with random values and trained from the scratch with a sufficient amount of dataset, the watermark can be removed completely without compromising the performance of DNN model. Hence, it is assumed in our method that the attacker cannot train the target DNN model from scratch in terms of computational resources and amount of training dataset.

The remainder of this chaper is organized as follows. Section 2 presents some assumptions of parameters. The proposed method is detailed in Section 3.3, and experimental results are presented in Section 3.4.

3.2 Conventional Works

In conventional work, the DM-QIM method [36, 37] is applied to reduce the effect on the parameters of the DNN model [38]. In [38], the loss function for embedding the watermark was omitted, assuming that the change in parameters in subsequent embedding operations would be less than the change in parameters in the first embedding operation. In [9], the amount of change due to the embedding was estimated by statistical analysis and proved to be minimal. To make it difficult to find the presence of watermark information, the watermark information is not directly embedded in the weights. First, n weights of the DNN models are randomly selected based on the secret key, and their frequency components are computed by discrete cosine transform (DCT). k DCT coefficients are selected from them. The weights are selected from the FC layers subject to fine-tuning. The procedure is described below.

1). Select n weights from FC layers according to a secret key key, which is denoted by a vector $\boldsymbol{f}$:

$$\boldsymbol{f} = (f_0, f_1, \ldots, f_{n-1}). \tag{3.1}$$

2). Perform DCT to the vector $\boldsymbol{f}$, and obtain the frequency components $\boldsymbol{F}$.

3). For each bit of watermark $\boldsymbol{w}$:

$$\boldsymbol{w} = (w_0, w_1, \ldots, w_{n-1}), \tag{3.2}$$

the corresponding frequency components F_i modified by using the DM-QIM method.

$$\overline{F_i} = \text{DM-QIM}(F_i, w_i, \delta, r_i), \tag{3.3}$$

where δ is the quantization step and r_i is the random dither signal generated by using a pseudo-random number generator PRNG ed from the range $[-\delta/2], \delta/2]$ and a secret key key.

$$\boldsymbol{r} = \text{PRNG}(key) = (r_0, r_1, \ldots, r_{k-1}) \tag{3.4}$$

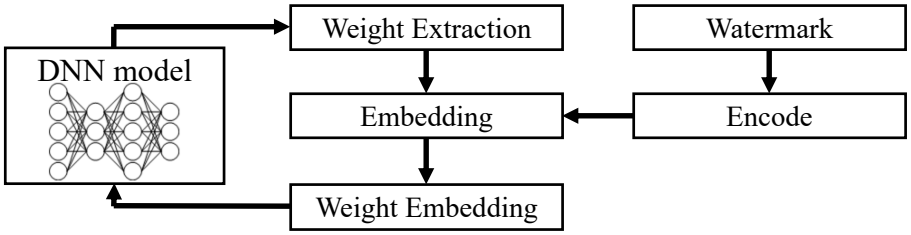


Figure 3.1: Flow diagram of embedding procedure.

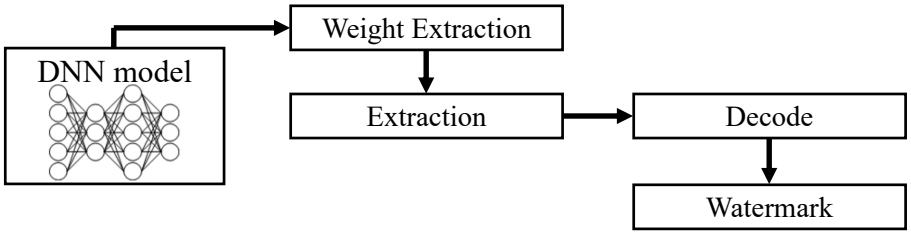


Figure 3.2: Flow diagram of extraction procedure.

4). Perform the inverse DCT to the vector $\overline{\boldsymbol{F}}$, and replace f_i with the watermarked weight $\overline{f_i}$ in the FC layers.

. Using the above method, the signal embedded as a watermark is spread over the sampled weights. One of the characteristics of the QIM method is that the distortion caused by the embedding is small. In addition, the introduction of secret keys makes it difficult to analyze the presence of a hidden message from observation of the weights in the FC layers. There is a trade-off between *Capacity* and *Robustness* against noise. However, it is difficult to add noise to eliminate the watermark with high accuracy, so this method is highly robust against noise. On the other hand, it is not robust against attacks that directly affect weights, such as pruning attacks.

3.3 Proposed DNN Watermarking

The overview of the proposed DNN watermarking is shown in Fig. 3.1 and Fig. 3.2, where Fig. 3.1 represents the embedding procedure and Fig. 3.2 represents the extraction procedure.

The idea is to encode the k-bit watermark $\boldsymbol{b}$ into the codeword $\boldsymbol{c}$ using Constant Weight Code(CWC) before the embedding operation.

3.3.1 Constant Weight Code

CWC $\mathcal{C}(\alpha, L)$ with parameters α and L is a set of binary codewords of length L, all having weight α; it has a fixed Hamming weight. Therefore, a codeword $\boldsymbol{c} = (c_0, c_1, \ldots, c_{L-1}), c_i \in \{0, 1\}$ of CWC satisfies the condition such that

$$\sum_{i=0}^{L-1} c_i = \alpha, \tag{3.5}$$

where α is fixed constant.

It is clear that no two codewords in $\mathcal{C}(\alpha, L)$ have Hamming distance 1 and the minimum distance is 2. This means that the code can detect only a single error and cannot correct any error at all. To make the CWC more practical, some researchers have been involved in developing a code with the restriction of a minimum distance d. The main problem in coding theory is finding the maximum possible number of codewords in a CWC with length L, minimum distance d, and weight α. Such CWCs have been extensively studied by MacWilliams and Sloane [46]. A large table of lower bounds on these numbers was published by Brouwer et al. [6], and it was updated by Smith et al. [63]. Because the CWC has no error-correction capability, even a 1-bit error is not allowed. Some studies have investigated CWCs with error-correcting capabilities [4, 20, 51]. These codes are a promising way to enhance the robustness against other attacks for DNN watermarking. Because of the simplicity, in our approach, we have adopted Schalkwijk's algorithm [59] for encoding and decoding the CWC with minimum distance 2, which does not restrict the use of other algorithms for constant weight code.

The encoding algorithm in [59] maps k-bit information into a codeword $\boldsymbol{c}$ with weight α and length L. The procedure to encode $\boldsymbol{b}$ into a codeword $\boldsymbol{c} \in \mathcal{C}(\alpha, L)$ is described in Algorithm 1. The k-bit information $\boldsymbol{b}$ is recovered by decoding the codeword $\boldsymbol{c}$ using Algorithm 2, where "$\gg$" denotes the right bit-shift operator.

The weights corresponding to the elements $c_i = 1$ becomes more than a higher threshold T_1, while the others corresponding to $c_i = 0$ becomes less than a lower threshold T_0 by the embedding operation. In case the pruning attack is executed to round weight parameters w_i with a small value to 0, those elements are judged as bit 0 in the codeword, and hence, there is no effect on the received codeword. As for bit 1, the corresponding weight parameters w_i should be sufficiently large so that these are not cut off.

Algorithm 1 Encode $\boldsymbol{b}$ into $\boldsymbol{c}$

Input: α, L, $\boldsymbol{b} = (b_0, b_1, \ldots, b_{k-1})$, $b_t \in \{0, 1\}$

Output: $\boldsymbol{c} = (c_0, c_1, \ldots, c_{L-1})$, $c_t \in \{0, 1\}$

$B \leftarrow \sum_{t=0}^{k-1} b_t 2^t$;
$\ell \leftarrow \alpha$;
for $t = 0$ **to** $L - 1$ **do**
  if $B \geq \binom{L-t-1}{\ell}$ **then**
    $c_{L-t-1} = 1$;
    $B \leftarrow B - \binom{L-t-1}{\ell}$;
    $\ell \leftarrow \ell - 1$;
  else
    $c_{L-t-1} = 0$;
  end if
end for

Algorithm 2 Decode $\boldsymbol{c}$ into $\boldsymbol{b}$

Input: α, L, $\boldsymbol{c} = (c_0, c_1, \ldots, c_{L-1})$, $c_t \in \{0, 1\}$

Output: $\boldsymbol{b} = (b_0, b_1, \ldots, b_{k-1})$, $b_t \in \{0, 1\}$

$B \leftarrow 0$;
$\ell \leftarrow 0$;
for $t = 0$ **to** $L - 1$ **do**
  if $c_t = 1$ **then**
    $\ell \leftarrow \ell + 1$;
    $B \leftarrow B + \binom{t}{\ell}$;
  end if
end for
for $t = 0$ **to** $k - 1$ **do**
  $b_t = B \pmod 2$
  $B \leftarrow B \gg 1$;
end for

3.3.2 Embedding

During initialization of a given DNN model, L weight parameters $\boldsymbol{w}$ are selected from N candidates according to a secret key. Then, an encoded watermark $\boldsymbol{c}$ is embedded into $\boldsymbol{w}$ under the following constraint:

- If $c_i = 1$, then $|w_i| \geq T_1$; otherwise, $|w_i| \leq T_0$, where T_0 and T_1 are thresholds satisfying $0 < T_0 < T_1$.

In the training process of a DNN model, weight parameters are updated iteratively to converge into a local minimum. The changes to the weights $\boldsymbol{w}$ selected for embedding $\boldsymbol{c}$ are only controlled by the above restriction during the training process in the proposed method. First, we encode a k-bit watermark $\boldsymbol{b}$ into the codeword $\boldsymbol{c}$ by using Algorithm 1. Here, the parameters α and L must satisfy the following condition:

$$2^k \leq \binom{L}{\alpha} = \frac{L!}{\alpha!(L-\alpha)!} < 2^{k+1}. \tag{3.6}$$

During the embedding operation, the weight parameters $\boldsymbol{w}$ selected from the DNN model are modified into $\boldsymbol{w}^{\dagger}$ by using the two thresholds T_1 and T_0.

$$w_i^{\dagger} = \begin{cases} w_i & (c_i = 1) \cap (|w_i| \geq T_1) \\ sgn(w_i) \cdot T_1 & (c_i = 1) \cap (|w_i| < T_1) \\ w_i & (c_i = 0) \cap (|w_i| \leq T_0) \\ sgn(w_i) \cdot T_0 & (c_i = 0) \cap (|w_i| > T_0) \end{cases}, \tag{3.7}$$

where

$$sgn(x) = \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases} \tag{3.8}$$

The embedding procedure is illustrated in Fig. 3.3.

Eq.(3.7) can be regarded as a constraint for executing the training process for the DNN model to embed the watermark. Among the numerous N candidates, we impose the constraint only on the L weight parameters selected for embedding.

3.3.3 Extraction

It is expected that the distribution of selected weights to be embedded is the same as the distribution of all candidates (Gaussian or uniform distribution). At the embedding a watermark, the change of the distribution depends on the thresholds T_1 and T_0 as well as the length L of encoded watermark.

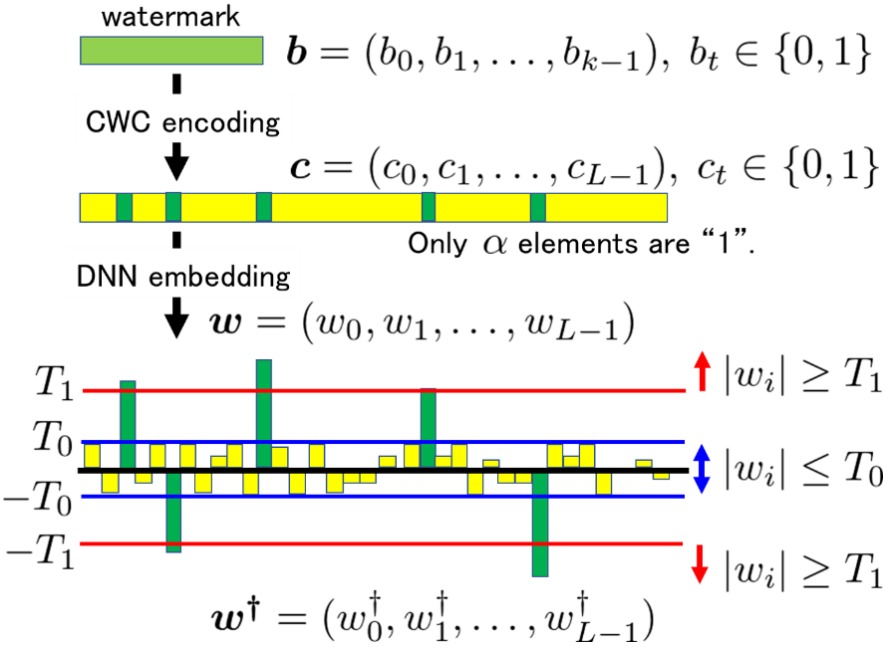


Figure 3.3: Illustration of embedding procedure.

3.3.3.1 Detection

To check the presence of watermark in a DNN model, it is sufficient to check the bias in the distribution of selected L weights. Here, if the secret key for selecting weights is known, the bias of the binary sequence of the selected weights can be used for checking the existence of watermark. Because the redHamming weight of the binary sequence is α and it is much smaller than L, the bias is useful to detect the existence of watermark. Under the assumption is that L is extremely small compared with all candidates, it is difficult, without the secret key, to find and change the selected weights under the constraint that the performance degradation of watermarked model is negligibly small.

If all weights in the DNN model are uniformly distributed, then the randomly selected weights are also uniformly distributed. To check whether the sequence of selected weights is the CWC codeword or not, we measure the mean square error (MSE) of the sequence. Suppose that the weights are selected from a uniform distribution with range $[0, \delta]$. If the sequence of selected weights is not a CWC codeword, the distribution is the uniform distribution in the range $[0, \delta]$ and the mean is $\delta/2$. On the other hand, if a CWC codeword is embedded, the distribution is different and is dependent on T_1 and T_0 as follows:

- Top α-th weights: the uniform distribution in the range $[T_1, \delta]$, the mean is $(\delta + T_1)/2$.
- Remainder: the uniform distribution in the range $[0, T_0]$, the mean is $T_0/2$.

We can determine whether the distribution of the sequence of weights is similar to the

CWC codeword or not. The MSE as the metrics is defined by the following equation.

$$MSE = \frac{1}{L}\sum_{i=0}^{L-1} d_i, \tag{3.9}$$

where

$$d_i = \begin{cases} (c_i - \frac{\delta+T_1}{2})^2 & (1 \le i < \alpha) \\ (c_i - \frac{T_0}{2})^2 & (\alpha \le i < L) \end{cases}. \tag{3.10}$$

3.3.4 Decode

First, the weight parameters are selected from the same DNN model positions, denoted by $\boldsymbol{w'}$. Then, the α-th largest element $\tilde{w}'_{L-\alpha}$ is determined from $\boldsymbol{w'}$, and the codeword $\boldsymbol{c'}$ is constructed as follows:

$$c'_i = \begin{cases} 1 & \text{if } |w'_i| \ge \tilde{w}'_{L-\alpha} \\ 0 & \text{otherwise} \end{cases}, \tag{3.11}$$

where $\tilde{\boldsymbol{w}}' = sort(|\boldsymbol{w}'|)$. Finally, using Algorithm 2, the watermark $\boldsymbol{b'}$ is reconstructed from the codeword $\boldsymbol{c'}$ as the result.

In the above operation, the top-α symbols in $\boldsymbol{w'}$ are regarded as "1," and the others are "0". Even if $L-\alpha$ symbols whose absolute value is smaller than that of the top-α symbols are pruned, the codeword can be correctly reconstructed from the weight parameters $\boldsymbol{w'}$ in the pruned DNN model. When the pruning rate R satisfies the condition

$$R < \frac{L-\alpha}{L} = \overline{R}, \tag{3.12}$$

statistically, no error occurs in the above extraction method. Because L weight parameters $\boldsymbol{w'}$ are sampled from N candidates in a DNN model for embedding the watermark, the above condition does not coincide with the actual robustness against a pruning attack with rate R.

3.3.5 Design of Two Thresholds

Because the weight of the codewords is constant, we selected α largest elements from L elements in the weight parameter $\boldsymbol{w'}$ extracted from the given DNN model. Although some weight parameters are cut off by the pruning attack, the values of such α elements can be retained if the threshold T_1 is appropriately designed.

Weight initialization is important to develop effective DNN models, and it is used to define the initial values for the parameters before training the models. The choice of Gaussian or uniform distribution does not have any effect, whereas the scale of the initial distribution has a significant effect on both the outcome of the optimization procedure

and the ability of the network to generalize [24]. When a Gaussian distribution is selected, whose variance is studied in [23], the method is referred to as Xavier initialization. Later, [27] revealed that the Xavier initialization method does not work well with the RELU activation function, and it was revised in [30] to accommodate non-linear activation functions. These studies are based on a Gaussian assumption for the initial values, and their variance can be calculated using the libraries of PyTorch and Tensorflow.

Here, we suppose that the value of weight parameters in a DNN model is modeled as a Gaussian distribution before and after the model's training. Because the pruning attack cuts off p weight parameters with small values, the absolute values of α elements in $\boldsymbol{w}^{\dagger}$ should be greater than them. A statistical analysis of the distribution gives us the following inequality for the threshold T_1. Fig. 3.4(a) shows the probability density function of the weight parameters. According to the figure, for a given threshold T_1, the pruning rate R can be calculated as

$$\begin{aligned} R &\leq \frac{1}{\sqrt{2\pi\sigma^2}} \int_{-T_1}^{T_1} \exp\Big(-\frac{x^2}{2\sigma^2}\Big) dx && (3.13) \\ &= 1 - \frac{2}{\sqrt{2\pi\sigma^2}} \int_{T_1}^{\infty} \exp\Big(-\frac{x^2}{2\sigma^2}\Big) dx && (3.14) \\ &= 1 - 2Q\Big(\frac{T_1}{\sigma}\Big), && (3.15) \end{aligned}$$

where $Q()$ is the Q-function

$$Q(t) = \frac{1}{\sqrt{2\pi}} \int_{t}^{\infty} \exp\Big(\frac{-x^2}{2}\Big) dx. \tag{3.16}$$

By using the inverse Q-function $Q^{-1}()$, the appropriate threshold T_1 can be calculated for a given pruning rate $\overline{R}$.

$$T_1 = \sigma Q^{-1}\Big(\frac{1-\overline{R}}{2}\Big) \tag{3.17}$$

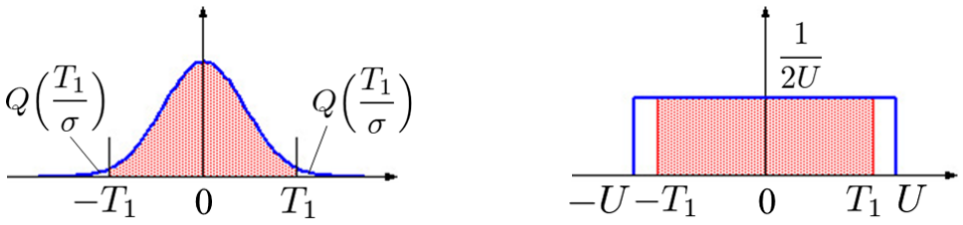
In the case of a uniform distribution in the range $[-U, U]$, the probability density function of the weight parameters is illustrated in Fig. 3.4(b); the shaded area in the figure corresponds to the rate R. Here, the threshold T_1 can be calculated to satisfy the condition:

$$R \leq 2T_1 \times \frac{1}{2U}. \tag{3.18}$$

Therefore, T_1 can be given as

$$T_1 = \overline{R}U. \tag{3.19}$$

For T_0, the condition $0 < T_0 < T_1$ is sufficient if only a pruning attack is assumed. Considering the robustness against other attacks that modify weight parameters in a watermarked model, we should consider an appropriate margin $T_1 - T_0$ by setting T_0. Setting a large value for T_1 or a small value for T_0 (a large margin of $T_1 - T_0$) is ideal in



(a) Gaussian (b) Uniform

Figure 3.4: Example of probability distribution function of initial weight values.

terms of robustness against pruning, but it increases the amount of change in the weights. This makes abnormal features appear in the distribution of the weights, thus exposing the watermark as an attack target. On the other hand, reducing the margin of $T_1 - T_0$ to prevent these features from appearing in the distribution of weights means increasing the possibility of bit-flip in the extracted codewords. Because these thresholds are a trade-off, it is necessary to set an appropriate margin of $T_1 - T_0$ according to requirements.

3.3.6 Considerations

For simplicity of explanation, we assume that the weight levels are pruned in the ascending order, starting from the weight with the smallest value. The watermarking method can be extended to support more advanced pruning methods in the selection of weights, considering the pruning criteria and settings discussed in [19, 48].

The usability of the proposed embedding method is confirmed from the studies of previous DNN watermarking methods. For instance, the constraint given by Eq.(3.7) can be applied to the embedding operations in [11, 50, 58, 68]. In the case of the method presented in [34], the embedding operation based on the constraint can be regarded as the initial assignment of weight parameters to a DNN model, and the change in weights at each epoch is corrected by iteratively performing the operation.

From the perspective of secrecy, it is better to select a small α. Attackers can use two possible approaches to cause a bit-flip in the codeword embedded into a DNN model. One approach is to identify the elements satisfying $|w_i| \geq T_1$ and decrease their weight values. Among the $N(1 - \overline{R})$ candidates of weight parameters $|w_i| \geq T_1$, identifying α values becomes difficult with the increase of N. Because the total number of weight parameters in a DNN model is very large, executing this approach is difficult without significantly changing the weight parameters in the model. The other approach is to increase the

weight values of selected weights whose values are $|w_i| \leq T_0$. Because of the large number of candidates, finding such weights without a secret key is challenging.

3.3.7 Numerical Examples

Table 3.1 enumerates some examples of parameters for CWC $\mathcal{C}(\alpha, L)$ with respect to bit length k of the watermark. For instance, when a 128-bit watermark is encoded with $\alpha = 20$, the length of its codeword becomes $L = 722$. Then, the code can withstand a pruning attack with a rate of $R < 0.9723$.

If the amount of watermark information is large, it is possible to divide it into small blocks and embed each encoded block into selected weight parameters without overlapping.

3.4 Experiments for Evaluations

In this section, we encode a watermark using CWC and then embed the codeword into DNN models to evaluate the effects on DNN models. The amount of watermark is fixed to $k = 128$ bits, and the codeword is generated by different combinations of α and L enumerated in Table 3.1. Ten different watermarks were selected from random binary sequences in this experiment. Then, the robustness against a pruning attack was measured by changing the rate R.

We considered the validity of the proposed method using accuracy and loss. In the case of binary classification, accuracy can be expressed as

$$Accuracy_{bin} = \frac{TP + TN}{TP + TN + FP + FN}, \tag{3.20}$$

where True Positive (TP) is a test result that correctly indicates the presence of a label, True Negative (TN) is a test result that correctly indicates the absence of a label, False Positive (FP) is a test result that wrongly indicates the presence of a particular label and False Negative (FN) is a test result that wrongly indicates the absence of a particular label, respectively. In the case of the multi-class classification used in our validation, accuracy is expressed as the average of the accuracy of each class, as shown below.

$$Accuracy = \frac{\sum_i^k Accuracy_i}{k}, \tag{3.21}$$

where, $Accuracy_i$ is i-class accuracy. And then, loss is calculated using categorical cross entropy as follows.

$$Loss = -\sum_t \sum_c y_c^{(t)} \log \hat{y}_c^{(t)}, \tag{3.22}$$

Table 3.1: Numerical examples of CWC parameters.

k	α	L	$\overline{R}$
64	8	972	0.9918
	9	583	0.9846
	10	393	0.9746
	11	288	0.9618
128	16	1757	0.9909
	18	1063	0.9831
	20	722	0.9723
	22	533	0.9587
256	32	3307	0.9903
	36	2011	0.9821
	40	1373	0.9709
	43	1090	0.9606
512	63	6858	0.9908
	73	3693	0.9802
	79	2780	0.9716
	85	2196	0.9613
1024	127	12955	0.9902
	145	7443	0.9805
	159	5350	0.9703
	170	4323	0.9607

where $y_c^{(t)}$ is the one-hot representation of the t-th training data, and $\hat{y}_c^{(t)}$ is the t-th model output. First, we compare the accuracy of the watermarked DNN model with that of the original DNN model for a given task. Second, we run the pruning attack and check the error rate, i.e., the ratio of the number of extraction failures to the number of trials. Finally, we evaluate the error rates of the pruned DNN model after retraining.

3.4.1 Experimental Conditions

We selected the VGG16 [61] and ResNet50 [28] models as pre-trained models. These models were trained using more than 1000000 images from the ImageNet [16] database. A watermark was embedded into the fine-tuning model during training, similar to the experiments in [34].

3.4.1.1 Fine-Tuning Model

Based on these pre-trained models, we fine-tuned the models with a batch size of 8 by replacing the new fully-connected (FC) layers connected to the final convolutional layer. The number of nodes at the final convolutional layer is 8192 ($= 4 \times 4 \times 512$) in VGG16 and 51200 ($= 5 \times 5 \times 2048$) in ResNet50, and these nodes are connected to new FC layers with 256 nodes. The number of candidates for selecting weights from the first FC layer is more than 2000000, $N = 8192 \times 256$, in VGG16. Similarly, it is more than 13000000, $N = 51200 \times 256$, in ResNet50. It is noted that the number N of weight parameters is much larger than the length L of the CWC codeword.

These fine-tuned models are trained using the 17 Category Flower Dataset[1] provided by the Visual Geometry Group of Oxford University—62.5% of images were used as training data, 12.5% as validation data, and 25.0% as test data.

In this experiment, two types of fine-tuning methods were used for different purposes: fine-tuning to embed the CWC codeword and retraining to reduce the effect of pruning attacks. The epochs for fine-tuning to embed are 50 for VGG16 and 100 for ResNet50, respectively. The number of epochs for retraining the pruned models is 5.

3.4.1.2 Threshold

The threshold T_1 must be designed appropriately to ensure robustness against pruning attacks. As discussed in Section 3.3.5, it depends strongly on the weight initialization. Owing to its simplicity, we selected the uniform distribution with the default setting of weight initialization [27] in the PyTorch environment.

In the VGG16-based model, we set the threshold $T_1 = 0.026$ for the uniform distribution in the range $[-0.026650, 0.026650]$. This indicates that the percentage of weight

[1] `https://www.robots.ox.ac.uk/~vgg/data/flowers/17/`

parameters whose values are smaller than the threshold T_1 is 97.56%; thus, $\overline{R} = T_1/U = 0.9756$. We also set the threshold $T_0 = T_1/2 = 0.013$. For the ResNet50-based model, we set the threshold $T_1 = 0.010$ for the uniform distribution in the range $[-0.010798, 0.010798]$. This indicates that the percentage of weight parameters with values smaller than the threshold T_1 is 92.61%; thus, $\overline{R} = 0.9261$. We also set the threshold $T_0 = T_1/2 = 0.005$.

3.4.2 Effect of Watermark

There should be no significant difference in accuracy between embedding a watermark into a DNN model and not embedding into it. For each Hamming weight of codewords α and its length L by $k = 128$ in Table 3.1, we compared DNN models with and without embedding in terms of accuracy and loss metrics, and the results are enumerated in Table 3.2, where we calculate the average of 10 trials in this experiment. Even though the results show some variation, the watermarked model does not show any noticeable difference from the original model. These results confirm that the effect of embedding a watermark into a DNN model on the performance is negligible. Although the original loss appears to be slightly higher than that of the model with embedding, this difference is within the margin of variation of the simulation.

3.4.3 Detection Performance

We have confirmed that the watermark embedded in the DNN model can be detected correctly. In the embedding procedure, weights are randomly selected from all weights in the DNN model based on a secret key. To detect the presence of hidden watermark, it is sufficient to determine whether this selected weights sequence is the CWC codeword or not.

We conducted the simulation under the setup of $\delta = 0.02665, T_1 = 0.026, T_0 = 0.013$. We generated 10000 codewords and 10000 non-codewords, respectively, where the codewords have code length $L = 1757$ and $\alpha = 16$. The non-codewords are a randomly selected sequence from a uniform distribution.

The MSE was calculated for each of the generated codewords and non-codewords. Fig. 3.5 shows the histogram of MSEs. As the figure shows, the distribution of MSEs can be clearly separated for codewords and non-codewords. This result implies that it is easy to distinguish codewords from non-codewords by setting a proper threshold.

3.4.4 Robustness Against Pruning Attacks

We measured the robustness of the CWC codeword against a pruning attack. We selected the threshold T_1 to ensure robustness against pruning attacks with a pruning rate

Table 3.2: Effect of embedding watermark when $k = 128$.

(a) VGG16

metric	phase	Original	$\mathcal{C}(16, 1757)$	$\mathcal{C}(18, 1063)$	$\mathcal{C}(20, 722)$	$\mathcal{C}(22, 533)$
accuracy	training	0.9639	0.9650	0.9648	0.9637	0.9634
	validation	0.9226	0.9137	0.9196	0.9238	0.9119
	test	0.9071	0.9041	0.9029	0.9088	0.9068
loss	training	0.1207	0.1147	0.1189	0.1175	0.1204
	validation	0.2288	0.2393	0.2398	0.2184	0.2575
	test	0.3703	0.3541	0.3614	0.3600	0.3662

(b) ResNet50

metric	phase	Original	$\mathcal{C}(16, 1757)$	$\mathcal{C}(18, 1063)$	$\mathcal{C}(20, 722)$	$\mathcal{C}(22, 533)$
accuracy	training	0.9926	0.9924	0.9925	0.9923	0.9923
	validation	0.9310	0.9310	0.9375	0.9304	0.9405
	test	0.9288	0.9326	0.9338	0.9300	0.9382
loss	training	0.0236	0.0251	0.0238	0.0246	0.0234
	validation	0.4501	0.4099	0.4581	0.4493	0.3696
	test	0.5039	0.5558	0.5410	0.5478	0.5192

of $\overline{R} = 0.9756$ and $\overline{R} = 0.9261$ for the VGG16-based and ResNet50-based models, respectively. Unfortunately, the distribution of weight parameters changed slightly after training. Therefore, we evaluated the robustness against pruning attacks by varying R in the range $[0, 0.9]$ in increments of 0.1.

In this evaluation, no error occurred in the extraction of the watermark. Therefore, for a detailed evaluation, we executed the pruning attack by varying the range $[0.9, 1.0)$ in increments of 0.01, whose results are shown in Table 3.3. The VGG16-based model has an error rate of 0% when the pruning rate is $R \leq 0.97$. The ResNet50-based model has no error for the pruning rate of $R \leq 0.92$. Thus, it is confirmed that the fine-tuned models based on VGG16 and ResNet50 are robust against pruning attacks if the pruning rate R is less than the designed rate $\overline{R}$, which can be determined using the CWC parameters α and L.

Note that robustness against a pruning attack whose pruning rate R is less than the designed pruning rate $\overline{R}$ is not always guaranteed, but it is statistically assured. This is because the thresholds are set from the distribution of the entire weight parameters in a DNN model, while the weights to be embedded are randomly selected from the entire weights based on the secret key. Nevertheless, under this condition, experiments show

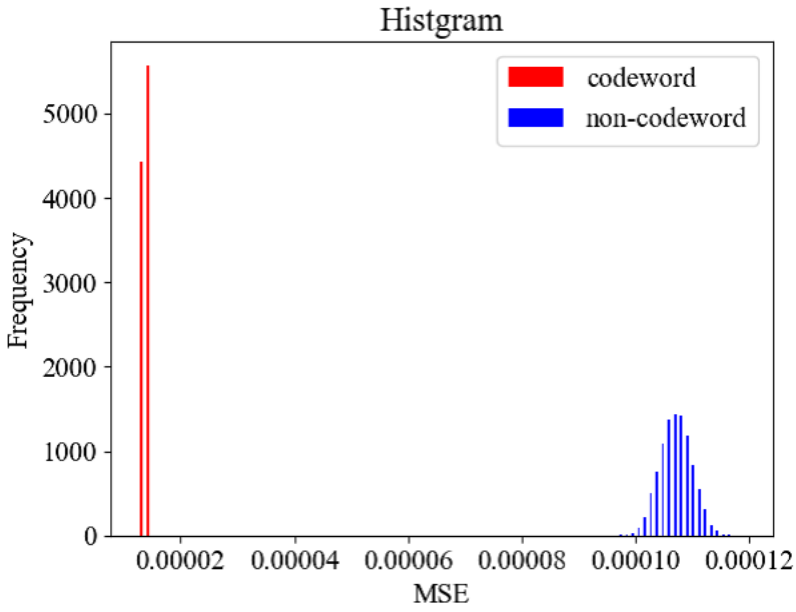


Figure 3.5: Histogram of MSEs of codeword and non-codeword.

that the robustness can be well-managed by carefully selecting the thresholds T_1 and T_0.

3.4.5 Retrained DNN Model After Pruning Attack

As mentioned in Section 2.2.2, the DNN model is retrained after the pruning attack to recover the accuracy of the original task. We measure the accuracy for the DNN models based on VGG16 and ResNet50 before and after retraining the pruned model. Without the retraining, the higher the pruning rate, the lower the accuracy for the VGG16-based method, while the ResNet50-based model seems to be less affected by pruning attacks. Among some possible hyper-parameters, the main difference between them will be the number of weight parameters. A detailed analysis will be performed in a future work.

Table 3.4 shows the error rate of the CWC codeword when the pruned models are retrained. It is observed that no error occurs in the VGG16-based model when the pruning rate is $R \leq 0.97$. This indicates that the watermark becomes robust against pruning attacks if $R < \overline{R}$. In the model using ResNet50, errors still occur even when the pruning rate is $R \leq 0.92$. We speculate that this is because the weights in the FC layers of ResNet50 are more sensitive to relearning and, thus, are more likely to change. This error can be avoided by embedding watermarks in the lower layers. Although the number of trials in this experiment is small, the results confirm that the CWC can be extracted even after the pruning attack and retraining. An extensive analysis will be performed in future work. These results demonstrate that encoding using CWC guarantees the robustness of a watermarked DNN model against pruning attacks, regardless of whether it has been retrained or not.

Table 3.3: Error rate against the pruning attack.

Base Model	code	Pruning Rate (R)								
		0.91	0.92	0.93	0.94	0.95	0.96	0.97	0.98	0.99
VGG-16	$\mathcal{C}(16, 1757)$	0	0	0	0	0	0	0	100	100
	$\mathcal{C}(18, 1063)$	0	0	0	0	0	0	0	100	100
	$\mathcal{C}(20, 722)$	0	0	0	0	0	0	0	100	100
	$\mathcal{C}(22, 533)$	0	0	0	0	0	0	0	100	100
ResNet-50	$\mathcal{C}(16, 1757)$	0	0	100	100	100	100	100	100	100
	$\mathcal{C}(18, 1063)$	0	0	100	100	100	100	100	100	100
	$\mathcal{C}(20, 722)$	0	0	100	100	100	100	100	100	100
	$\mathcal{C}(22, 533)$	0	0	100	100	100	100	100	100	100

3.4.6 Comparison with previous studies

Table 3.5 shows the effect of attacks on performance when ascending pruning attacks and random pruning attacks are performed with increasing pruning rates in previous studies [41, 68, 71]. In the ascending pruning attack, the top R % parameters are cut-off according to their absolute values in ascending, while in the random pruning attack, R % of parameters are randomly removed. In the evaluation of multilayer perceptron (MLP) and VGG, the bit error rate (BER) is zero up to a pruning rate of 0.9; in the evaluation of Wide ResNet (WRN), the BER is zero up to a pruning rate of 0.6 or 0.65. VGG/RN is our proposed method. As discussed in Section 3.3.5, the robustness of our method can be controlled by defining the pair of code parameters α and L, and an appropriate threshold T_1, which makes our method more robust than the existing methods.

Table 3.4: Error rate against pruning attacks after retraining.

Base Model	code	Pruning rate (R)								
		0.91	0.92	0.93	0.94	0.95	0.96	0.97	0.98	0.99
VGG-16	$\mathcal{C}(16, 1757)$	0	0	0	0	0	0	10	100	100
	$\mathcal{C}(18, 1063)$	0	0	0	0	0	0	0	100	100
	$\mathcal{C}(20, 722)$	0	0	0	0	0	0	0	100	100
	$\mathcal{C}(22, 533)$	0	0	0	0	0	0	0	100	100
ResNet-50	$\mathcal{C}(16, 1757)$	90	90	100	100	100	100	100	100	100
	$\mathcal{C}(18, 1063)$	90	90	100	100	100	100	100	100	100
	$\mathcal{C}(20, 722)$	90	70	90	100	100	100	100	100	100
	$\mathcal{C}(22, 533)$	70	60	60	100	100	100	100	100	100

Table 3.5: Comparison of the performances of existing methods.

baseline model	Ascending	Random
MLP/VGG [71]	0.90	0.90
WRN [68]	0.65	0.65
WRN [41]	-	0.60
VGG/RN	0.97/0.92	-

Chapter 4

Near-Optimal Detection for Binary Tardos Code by Estimating Collusion Strategy

4.1 Introduction

In this chapter, we have developed an effective estimator for these parameters that uses the characteristics of the discretized probabilistic distribution of the Nuida code. This estimator has two steps.

In the first step, the estimator observes the bias of the "1" symbols in the pirated codeword and then forms a feature vector in accordance with the characteristics of the bias-based fingerprinting code. Essentially, each symbol in the codeword is determined by each assigned bias probability as a secret sequence. Therefore, the bias of symbol "1" in each innocent user codeword is statistically stable and depends only on the bias probability. In contrast, the bias in the pirated codeword differs as it is affected by the collusion strategy and number of colluders. Because the number of candidates for the bias probability in the Nuida code is finite, the symbols in the pirated codeword can be classified into groups having the same bias probabilities. The expected probability of each symbol in a group becoming 1 after a collusion attack is calculated. For each collusion strategy and number of colluders, almost all sets of expected probabilities are different. For convenience, such a set is defined as a Collusion Strategy Characteristic Vector (CSCV).

In the second step, the estimator identifies the CSCV closest to the feature vector, i.e., the one at a minimum distance from the CSCV, and estimates the collusion strategy and number of colluders. The use of this technique enabled a detector to achieve estimation accuracy greater than 90% against seven well-known collusion strategies and the other three collusion strategy combinated with well-known collusion strategies, with

near-optimal traceability.

We also investigated a noisy case representing a realistic scenario [31,32]. A codeword embedded in multimedia content using a watermarking technique was considered. If a pirated copy is produced by a coalition of illegal users, the codeword is further modified by signal processing operations such as lossy compression and filtering. This results in the addition of noise to the pirated codeword. As a result, the number of "1" and "0" symbols cannot be derived directly from the codeword. Thus, an additional estimator is required to adjust the parameters. The experimental results show that the traceability of the proposed method is still very near to optimal in the presence of noise.

Furthermore, we try to reduce the dimension of features in CSCVs for understanding the dominant features. In the experiment, we evaluated the accuracy in a different given maximum number of colluders.

The remainder of this chapter is organized as follows. Related studies are then discussed in Section 4.2. Section 4.3 introduces three proposed estimators. A reduction of dimension is mentioned in Section 4.4, and the experimental results are presented in Section 4.5.

4.2 Scoring Functions for Detection

Unfortunately, the scoring function uses only half of the information about the pirated codeword because the value of score $S_{j,i}$ is zero when $y_i = 0$. To use all of the information, Škorić et al. [70] proposed using the symmetric version of the scoring function:

$$S_j^{sym} = \sum_{i=1}^{L} S_{j,i}^{sym} = \sum_{i=1}^{L} (2y_i - 1)U_{j,i}. \tag{4.1}$$

This function requires no information about the collusion strategy or the number c of colluders. To discriminate colluders from innocent users, an optimal scoring function should be designed using these parameters from an information theoretical point of view. Furon and Perez-Freire defined the optimal scoring function for a single detector as a log-likelihood ratio [21]:

$$S_j^{MAP} = \sum_{i=1}^{L} S_{j,i}^{MAP} = \sum_{i=1}^{L} \log\left(\frac{\Pr[y_i|x_{j,i}, \boldsymbol{\theta_c^{str}}]}{\Pr[y_i|\boldsymbol{\theta_c^{str}}]}\right), \tag{4.2}$$

where a single detector computes a score for each user while a joint detector computes a score for a subset of users. As this score represents the maximum a posteriori probability, the optimal scoring function is called the MAP detector. The denominator $\Pr[y_i|p_i, \boldsymbol{\theta_c^{str}}]$

can be calculated using

$$\begin{cases} \Pr[1|\boldsymbol{\theta}_{\boldsymbol{c}}^{\boldsymbol{str}}] = \sum_{\lambda=0}^{c} \theta_{\lambda}^{str} \binom{c}{\lambda} p_i^{\lambda} (1-p_i)^{c-\lambda} \\ \Pr[0|\boldsymbol{\theta}_{\boldsymbol{c}}^{\boldsymbol{str}}] = 1 - \Pr[1|\boldsymbol{\theta}_{\boldsymbol{c}}^{\boldsymbol{str}}] \end{cases}. \tag{4.3}$$

Similarly, the numerator $\Pr[y_i|x_{j,i}, p_i, \boldsymbol{\theta}_{\boldsymbol{c}}^{\boldsymbol{str}}]$ can be calculated using

$$\begin{cases} \Pr[1|1, \boldsymbol{\theta}_{\boldsymbol{c}}^{\boldsymbol{str}}] = \sum_{\lambda=1}^{c} \theta_{\lambda}^{str} \binom{c-1}{\lambda-1} p_i^{\lambda-1} (1-p_i)^{c-\lambda} \\ \Pr[0|1, \boldsymbol{\theta}_{\boldsymbol{c}}^{\boldsymbol{str}}] = 1 - \Pr[1|1, \boldsymbol{\theta}_{\boldsymbol{c}}^{\boldsymbol{str}}] \\ \Pr[1|0, \boldsymbol{\theta}_{\boldsymbol{c}}^{\boldsymbol{str}}] = \sum_{\lambda=0}^{c-1} \theta_{\lambda}^{str} \binom{c-1}{\lambda} p_i^{\lambda} (1-p_i)^{c-\lambda-1} \\ \Pr[0|0, \boldsymbol{\theta}_{\boldsymbol{c}}^{\boldsymbol{str}}] = 1 - \Pr[1|0, \boldsymbol{\theta}_{\boldsymbol{c}}^{\boldsymbol{str}}] \end{cases}. \tag{4.4}$$

Moulin studied the theoretical aspect of a joint detector [49], and Meerwald and Furon proposed a practical implementation [47] that can be extended from a single detector. Therefore, we focus on a single detector here. Both theoretically and practically, the difficulty in designing such an optimal scoring function is how to estimate the collusion strategy *str* and the number of colluders c, namely $\boldsymbol{\theta}_{\boldsymbol{c}}^{\boldsymbol{str}}$, from a given codeword $\boldsymbol{y}$. Furon and Perez-Freire estimated these parameters using an expectation-maximization (EM) algorithm [21], but the accuracy of this approach is not high. To the best of our knowledge, there have been no other studies of the estimator.

We first discuss the universal scoring function, which achieves better performance for an arbitrary collusion strategy than uninformed methods such as Škorić's symmetric scoring function [70]. Because of the difficulty of realizing the MAP detector, the scoring function has been adjusted so that a certain collusion strategy can achieve universality [1, 18, 39, 47, 55]. Bias in symbols "0" and "1" is observed, and the weights corresponding to the biases in Škorić's scoring function are used to calculate the score [33]. In this section, we review two scoring functions for our proposed estimator.

4.2.1 Bias Equalizer

In binary fingerprinting codes, the number of "0" and "1" symbols is balanced because of the symmetry of bias probability p_i. However, this balance is not always achieved in a pirated codeword. Škorić's scoring function can be modified to compensate for the imbalance created by a collusion attack by equalizing the balance using weighting parameters, giving a "bias equalizer" [33]. Let $\mathcal{Y}_1$ and $\mathcal{Y}_0$ be the set of indices i satisfying $y_i = 1$ and $y_i = 0$, respectively. Then, the numbers of elements in $\mathcal{Y}_1$ and $\mathcal{Y}_0$ are denoted by L_1 and L_0, respectively, where $L_1 + L_0 = L$. Because of the symmetry of a bias distribution,

it is expected that $L_1 = L_0$ unless the colluders do not know the actual values $x_{j,i}$ of their codewords. Therefore, in the case of $\boldsymbol{y}$ produced by "all-0" and "all-1," L_1 is not always equal to L_0. As mentioned in Section II-B, each probability p_i can be classified into ξ groups. The number of elements in the ξ-th group is denoted by ℓ_ξ, where $\ell_\xi \geq 0$ and $\sum_{\xi=1}^{n_g} \ell_\xi = L$. Additionally, the number of "1" and "0" symbols are denoted by $\ell_{\xi,1}$ and $\ell_{\xi,0}$, respectively. Note that $\ell_{\xi,1} + \ell_{\xi,0} = \ell_\xi$. As an example, when $c_{max} = 8$, the classification of $n_g = 4$ groups is illustrated in Fig. 4.1. Using those parameters, the scoring function in the bias equalizer is as follows.

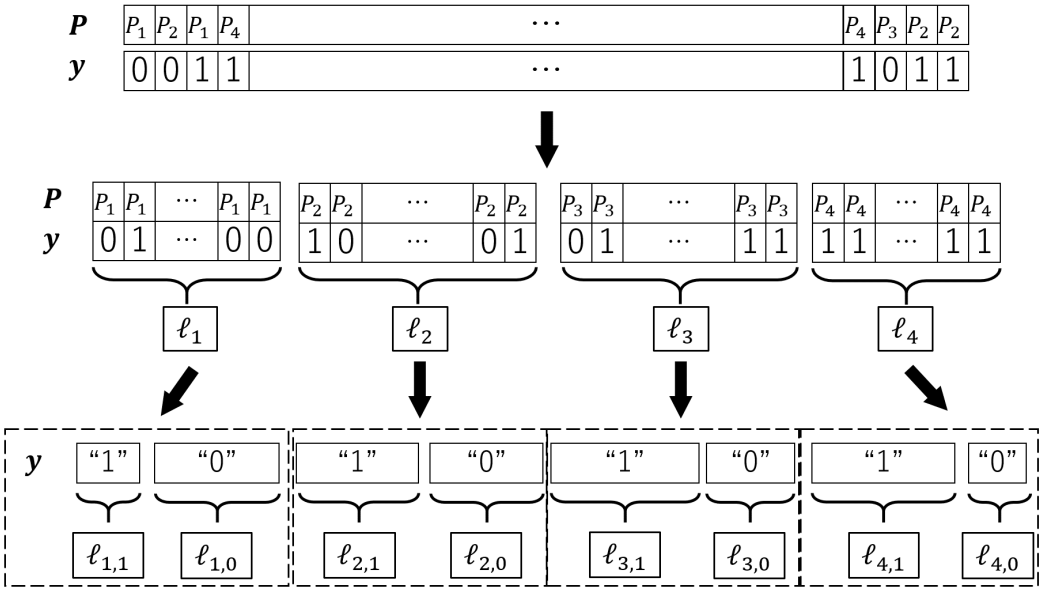


Figure 4.1: Number of "0" and "1" symbols in pirated codeword.

$$S_{j,i,\xi}^{Bias} = y_i \begin{cases} U_{j,i}^{00} = \dfrac{\ell_{\xi,1}}{\ell_\xi}\sqrt{\dfrac{p_i}{1-p_i}} & (x_{j,i} = y_i = 0) \\ U_{j,i}^{01} = -\dfrac{\ell_{\xi,0}}{\ell_\xi}\sqrt{\dfrac{p_i}{1-p_i}} & (x_{j,i} = 1, y_i = 0) \\ U_{j,i}^{10} = -\dfrac{\ell_{\xi,1}}{\ell_\xi}\sqrt{\dfrac{1-p_i}{p_i}} & (x_{j,i} = 0, y_i = 1) \\ U_{j,i}^{11} = \dfrac{\ell_{\xi,0}}{\ell_\xi}\sqrt{\dfrac{1-p_i}{p_i}} & (x_{j,i} = y_i = 1) \end{cases} \tag{4.5}$$

To adjust the above weighting parameters on the basis of the gap for each collusion strategy, the collusion strategy is classified into three types (all-0 or all-1 attack, minority or coin-flip attack, or other) for the bias equalizer [33]. First, the conditions in Eq. (4.6) were identified.

$$\begin{cases} \ell_{\xi,0} \approx \ell_\xi, & \text{if } p_i < 0.5 \text{ holds for all } \xi \\ \ell_{\xi,1} \approx \ell_\xi, & \text{if } p_i > 0.5 \text{ holds for all } \xi \end{cases} \tag{4.6}$$

All-0, all-1, and other strategies can be classified by introducing a threshold $T^{\dagger}$. For the classification of all-0 and all-1 attacks, the following two cases are checked.

$$\begin{cases} \dfrac{\ell_{\xi,0}}{\ell_{\xi}} > T^{\dagger} & (p_i < 0.5) \\ \dfrac{\ell_{\xi,1}}{\ell_{\xi}} > T^{\dagger} & (p_i > 0.5) \end{cases} \tag{4.7}$$

Note that $T^{\dagger}$ is close to 1 because of Eq. (4.6). In a previous study [33], threshold $T^{\dagger}$ was empirically determined to be 0.95. When the minority or coin-flip attack strategy is used, the following relations can be observed for the ξ-th group.

$$\begin{cases} \dfrac{\ell_{\xi,0}}{\ell_{\xi,1}} < \sqrt{\dfrac{1-p_i}{p_i}} & (p_i < 0.5) \\ \dfrac{\ell_{\xi,1}}{\ell_{\xi,0}} < \sqrt{\dfrac{p_i}{1-p_i}} & (p_i > 0.5) \end{cases} \tag{4.8}$$

Finally, a pirated codeword that has passed the above two steps is classified as being generated by one of the strategies used in majority, interleave, and worst case attacks. In this case, the collusion strategies are classified into one of three types, and an improved scoring function is used to revise the weights. Even though the bias equalizer improves the performance over that of uninformed scoring functions such as the symmetric decoder [70], the classification of collusion strategies is heuristic rather than theoretical. It is thus necessary to study a theoretical derivation for estimating collusion strategies.

4.2.2 Estimating Number of Colluders

Information about the number of colluders is also required for scoring functions based on the MAP detector. For example, Meerwald et al. [47] assumed that the number of colluders is less than or equal to c_{max} and calculated the correlation scores for the number of colluders within $[1, c_{max}]$ for a scoring function based on the MAP detector. When the function is adjusted for WCA $\boldsymbol{\theta}_{\boldsymbol{\lambda}}^{\boldsymbol{WCA}}(1 \leq \lambda \leq c_{max})$, score $S_{j,i}^{WCA}$ is determined by the candidate with the maximum value.

$$S_j^{WCA} = \max_{1 \leq \lambda \leq c_{max}} \left(\sum_{i=1}^{L} \left(\log \frac{\Pr[y_i | x_{j,i}, \boldsymbol{\theta}_{\boldsymbol{\lambda}}^{\boldsymbol{WCA}}]}{\Pr[y_i | \boldsymbol{\theta}_{\boldsymbol{\lambda}}^{\boldsymbol{WCA}}]} \right) \right). \tag{4.9}$$

This scoring function is called WCA defense because the score in Eq. (4.2) is oriented for a WCA attack. The method first calculates the c_{max} scores, from which the final score is produced. Thus, the number of colluders c is not directly estimated. Meerwald et al. also proposed a maximum likelihood estimator that guesses the collusion strategy $\boldsymbol{\theta}$ from a given pirated codeword.

4.3 Proposed Estimator

This section describes the proposed estimator to estimate vector $\boldsymbol{\theta}_{\boldsymbol{c}}^{\boldsymbol{str}}$ for the optimal MAP detector. We exploit the bias in a pirated codeword to generate the estimate.

4.3.1 Collusion Strategy Characteristic Vector

When a pirated codeword is produced by a combination of codewords under the constraint of the marking assumption, the number of "0" and "1" symbols must have changed. We measure the number of changes on the basis of the discrete bias probability. The emerging probability P_ξ, $(1 \leq \xi \leq n_g)$ is statistically equivalent to $\ell_{\xi,1}/\ell_\xi$ for each user's codeword. Hence, if we observe the number of symbols in a codeword, the following condition must be satisfied:

$$(P_1, \ldots, P_\xi, \ldots, P_{n_g}) \approx \left(\frac{\ell_{1,1}}{\ell_1}, \ldots, \frac{\ell_{\xi,1}}{\ell_\xi}, \ldots, \frac{\ell_{n_g,1}}{\ell_{n_g}} \right). \tag{4.10}$$

The right-hand term in Eq. (4.10) will be changed in a pirated codeword, and the number of changes in each element depends on the collusion strategy and the number of colluders. For convenience, the vector observed from a pirated codeword is denoted by

$$\boldsymbol{\Gamma} = (\gamma_1, \ldots, \gamma_\xi, \ldots, \gamma_{n_g}), \tag{4.11}$$

where

$$\gamma_\xi = \frac{\ell_{\xi,1}}{\ell_\xi}. \tag{4.12}$$

The expectation of the elements in $\boldsymbol{\Gamma}$ can be calculated from $\boldsymbol{\theta}_{\boldsymbol{c}}^{\boldsymbol{str}}$ and c as

$$\gamma_{c,\xi}^{str} = \sum_{\lambda=0}^{c} \binom{c}{\lambda} P_\xi^\lambda (1 - P_\xi)^{c-\lambda} \theta_\lambda^{str}. \tag{4.13}$$

The vector $\boldsymbol{\Gamma}_{\boldsymbol{c}}^{\boldsymbol{str}} = (\gamma_{c,1}^{str}, \ldots, \gamma_{c,\xi}^{str}, \ldots, \gamma_{c,n_g}^{str})$ is called the CSCV. Under the marking assumption, Eq. (4.13) enables us to express $\boldsymbol{\Gamma}_{\boldsymbol{c}}^{\boldsymbol{str}}$ for every general collusion strategy we can conceive. Several example collusion strategies are listed in Table 4.1, where $c_{max} = 8$ for the Nuida code and the actual number of colluders c is 6. We store the CSCVs $\boldsymbol{\Gamma}_{\boldsymbol{c}}^{\boldsymbol{str}}(\tilde{c}_{min} \leq c \leq \tilde{c}_{max})$ with the general collusion strategy into a database, where $\tilde{c}_{min}$ and $\tilde{c}_{max}$ are the assumed minimum and maximum number of colluders, respectively. Apart from these thresholds, we can measure the distance from feature vector $\boldsymbol{\Gamma}$ to each CSCV and find the closest $\boldsymbol{\Gamma}_{\boldsymbol{c}}^{\boldsymbol{str}}$ for each possible collusion strategy.

4.3.2 Vector Space

We define a vector space $\mathbb{Z}_2^L$ for a codeword represented by a binary vector of length L. The calculation of CSCV $\boldsymbol{\Gamma}_{\boldsymbol{c}}^{\boldsymbol{str}}$ from a given pirated codeword can be regarded as the

Table 4.1: Collusion strategy characteristic vectors for $c = 6$.

$\boldsymbol{\Gamma}_{\mathbf{6}}^{\boldsymbol{str}}$	ξ			
	$\gamma_{6,1}^{str}$	$\gamma_{6,2}^{str}$	$\gamma_{6,3}^{str}$	$\gamma_{6,4}^{str}$
$\boldsymbol{\Gamma}_{\mathbf{6}}^{\boldsymbol{maj}}$	0.00301	0.20498	0.79502	0.99699
$\boldsymbol{\Gamma}_{\mathbf{6}}^{\boldsymbol{min}}$	0.34762	0.70586	0.29414	0.65238
$\boldsymbol{\Gamma}_{\mathbf{6}}^{\boldsymbol{coin}}$	0.17531	0.45542	0.54458	0.82469
$\boldsymbol{\Gamma}_{\mathbf{6}}^{\boldsymbol{all0}}$	0.00000	0.00129	0.09045	0.64937
$\boldsymbol{\Gamma}_{\mathbf{6}}^{\boldsymbol{all1}}$	0.35063	0.90955	0.99871	1.00000
$\boldsymbol{\Gamma}_{\mathbf{6}}^{\boldsymbol{int}}$	0.06943	0.33001	0.66999	0.93057
$\boldsymbol{\Gamma}_{\mathbf{6}}^{\boldsymbol{WCA}}$	0.16699	0.34689	0.65311	0.83301

mapping from vector space $\mathbb{Z}_2^L$ to a rational vector space with n_g dimension $\mathbb{R}^{n_g}$. The CSCV's $\boldsymbol{\Gamma}_{\boldsymbol{c}}^{\boldsymbol{str}}$ can be derived from the following reduced-dimension map f in accordance with $L \gg n_g$.

$$f : \mathbb{Z}_2^L \mapsto \mathbb{R}^{n_g} \tag{4.14}$$

The symmetric decoder computes the correlation score between the pirated codeword and the users' codewords. As mentioned in Section 2.3.3, the realization of the MAP detector depends on how we estimate the collusion strategy and the number of colluders. Therefore, the map Eq. (4.14) implies that the detection of colluders can be performed in a lower dimension.

4.3.3 Noiseless Case

4.3.3.1 Basic Method

Let $D^{str,c}$ be the distance between the observed vector $\boldsymbol{\Gamma}$ and all CSCVs $\boldsymbol{\Gamma}_{\boldsymbol{c}}^{\boldsymbol{str}}$. Note that $\boldsymbol{\Gamma}_{\boldsymbol{c}}^{\boldsymbol{str}}$ can be calculated using Eq. (4.13) in advance and stored in a database. The basic method finds the closest vector $\boldsymbol{\Gamma}_{\boldsymbol{c}}^{\boldsymbol{str}}$ that minimizes distance D_c^{str}.

Well-known metrics for distance are the Total Variation distance D_1, the Euclidean distance D_2, and the Hellinger distance D_{Hel}:

$$D_1^{str,c} = \sum_{\xi} |\gamma_{c,\xi}^{str} - \gamma_{\xi}|, \tag{4.15}$$

$$D_2^{str,c} = \sqrt{\sum_{\xi} \left(\gamma_{c,\xi}^{str} - \gamma_{\xi}\right)^2}. \tag{4.16}$$

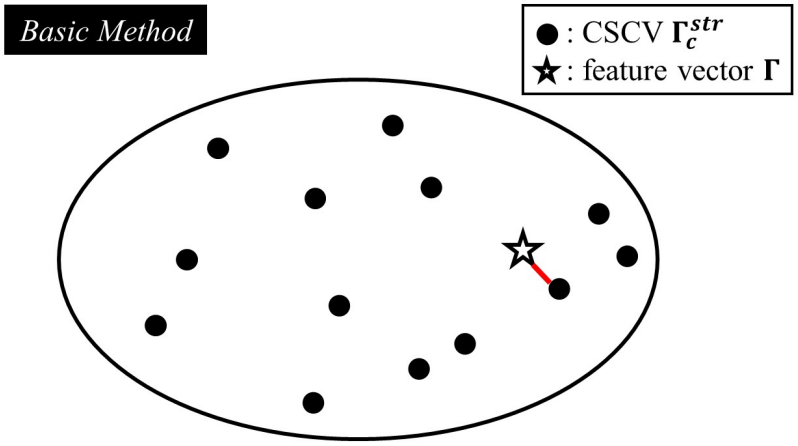


Figure 4.2: Illustration of vector space $\mathbb{R}^{n_g}$ and estimation process in basic method.

$$D_{Hel}^{str,c} = \sqrt{\sum_{\xi} \left(\sqrt{\gamma_{c,\xi}^{str}} - \sqrt{\gamma_{\xi}} \right)^2}. \tag{4.17}$$

In estimating the collusion strategy and number of colluders, we calculate $D^{str,c}$ for all CSCVs and find the combination that minimizes $D^{str,c}$ in vector space $\mathbb{R}^{n_g}$:

$$(\tilde{str}, \tilde{c}) = \arg\min_{str,c} D^{str,c}. \tag{4.18}$$

Fig. 4.2 shows vector space $\mathbb{R}^{n_g}$ and illustrates the estimation process in the basic method. The user's score S_j^{basic} is then calculated using

$$S_j^{basic} = \sum_{i=1}^{L} S_{i,j}^{basic} = \sum_{i=1}^{L} \log \left(\frac{\Pr[y_i | x_{j,i}, \boldsymbol{\theta}_{\tilde{c}}^{\tilde{str}}]}{\Pr[y_i | \boldsymbol{\theta}_{\tilde{c}}^{\tilde{str}}]} \right). \tag{4.19}$$

4.3.3.2 Subset Method

The proposed estimator searches for possible collusion attacks using the CSCVs stored in a database using a somewhat exhaustive search. Instead of a fully exhaustive search, the subset method calculates a set of user scores for some candidate number of colluders and outputs the number that maximizes the score.

The vector space $\mathbb{R}^{n_g}$ of all CSCVs is first separate into c subsets. A candidate CSCV can then be estimated in the subset. Fig. 4.3 illustrates the estimation process. An improvement in the estimation accuracy can be expected because the number of candidates $\tilde{c}_{max} - \tilde{c}_{min}$ in the subsets is reduced. Finally, the candidate user scores $\tilde{S}_{j,i}^{c}$ are calculated for estimated collusion strategy $\tilde{str}$ and for the number of colluders c corresponding to its subset. The maximum score is then determined as the user's score S_j^{sub}. This process is summarized as follows:

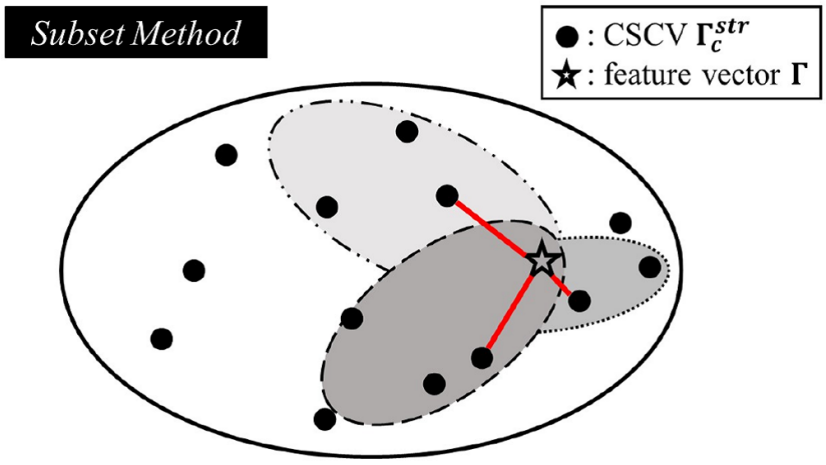


Figure 4.3: Illustration of estimation process in subset method.

1. Initialize $c = \tilde{c}_{min}$.

2. Perform the following operations until $c = \tilde{c}_{max}$.

   2-1) Estimate strategy $\tilde{str}$ using Eq. (4.18) by fixing $\tilde{c} = c$.

   2-2) Increment $c = c + 1$.

   2-3) Using estimated vector $\boldsymbol{\theta}_{\boldsymbol{c}}^{\tilde{\boldsymbol{str}}}$, calculate $S_{j,i}^{c}$ for $1 \leq i \leq L$ as

$$S_{j,i}^{c} = \log\left(\frac{\Pr[y_i|x_{j,i}, \boldsymbol{\theta}_{\boldsymbol{c}}^{\tilde{\boldsymbol{str}}}]}{\Pr[y_i|\boldsymbol{\theta}_{\boldsymbol{c}}^{\tilde{\boldsymbol{str}}}]}\right). \tag{4.20}$$

3. Calculate total score S_j^{sub} by summarizing maximum scores $S_{j,i}^{c}$ for $1 \leq i \leq L$.

$$S_j^{sub} = \max_{\tilde{c}_{min} \leq \lambda \leq \tilde{c}_{max}} \left(\sum_{i=1}^{L} S_{j,i}^{\lambda}\right) \tag{4.21}$$

As $\tilde{c}_{max} - \tilde{c}_{min}$ increases, the computational cost increases linearly because step 2 is repeated $\tilde{c}_{max} - \tilde{c}_{min}$ times. For example, if $\tilde{c}_{min} = 2$ and $\tilde{c}_{max} = 10$, the computational cost of the subset method is nine times greater than that of the basic method. However, the subset method achieves higher estimation accuracy of the collusion strategy in each subset because the number of candidates for estimation in a subset is fewer than in the basic method's set.

4.3.3.3 Dynamic Method

A preliminary experiment showed that the number of colluders detected by the subset method is greater than in the basic method though the computational cost is proportional

to the number of candidate vectors $\boldsymbol{\theta}_{\boldsymbol{c}}^{\tilde{\boldsymbol{str}}}$. In short, there is a trade-off between computational cost and traceability. Hence, we consider a new method that changes the number of $\boldsymbol{\theta}_{\boldsymbol{c}}^{\tilde{\boldsymbol{str}}}$ dynamically while maintaining detection accuracy. This is called the dynamic method.

In vector space $\mathbb{R}^{n_g}$, we introduce an (n_g-1)-hypersphere $\Omega^{n_g-1} = \{\boldsymbol{z} \in \mathbb{R}^{n_g} : \|\boldsymbol{z}\| = d\}$, where the radius d is a given positive number. For $n_g = 2$ and $n_g = 3$, the 1-hypersphere Ω^1 and 2-hypersphere Ω^2 are called a circle and a sphere, respectively. As mentioned in Section 4.3.2, any CSCV can be expressed by points in vector space $\mathbb{R}^{n_g}$, and the vector $\boldsymbol{\Gamma}$ derived from a pirated codeword can be placed at a certain point in $\mathbb{R}^{n_g}$. When we consider the (n_g-1)-hypersphere Ω^{n_g-1} centered at a point in $\boldsymbol{\Gamma}$, one of the CSCV candidates around $\boldsymbol{\Gamma}$ should have a high probability of being correct. The CSCV candidates for vector space $\mathbb{R}^{n_g}$ are illustrated in Fig. 4.4. The score in the dynamic method is calculated as follows:

1. Calculate distances $D^{str,c}$ for all possible CSCVs.
2. Form a set $\mathcal{D}$ of pairs (str, c) for which $D^{str,c}$ is less than d.
3. Perform the following operations if $\mathcal{D} \neq \{\text{null}\}$.

   3-1) Calculate scores $S_{j,i}^{str,c}$ with $\boldsymbol{\theta}_{\boldsymbol{c}}^{\boldsymbol{str}}$ for all pairs $(str, c) \in \mathcal{D}$.

   $$S_{j,i}^{str,c} = \log\left(\frac{\Pr[y_i|x_{j,i}, \boldsymbol{\theta}_{\boldsymbol{c}}^{\boldsymbol{str}}]}{\Pr[y_i|\boldsymbol{\theta}_{\boldsymbol{c}}^{\boldsymbol{str}}]}\right) \tag{4.22}$$

   3-2) Calculate total score $S_j^{dynamic}$ by summing maximum scores $S_{j,i}^{str,c}$ for $1 \leq i \leq L$.

   $$S_j^{dynamic} = \max_{(str,c)\in\mathcal{D}}\left(\sum_{i=1}^{L} S_{j,i}^{str,c}\right) \tag{4.23}$$

4. If $\mathcal{D} = \{\text{null}\}$, $S_j^{dynamic} = S_j^{basic}$.

4.3.4 Noisy Case

In practical situations, the pirated codeword may be distorted by noise. Noise can be modeled as additive white Gaussian noise (AWGN) [32]. This section shows how we estimate the collusion strategy and the number of colluders using CSCVs from a pirated codeword distorted by AWGN.

First, an EM algorithm is applied to all symbols of the pirated codeword for estimating noise variance σ_e^2. Then, γ_ξ is estimated from the symbols in the ξ-th group by using the estimated variance.

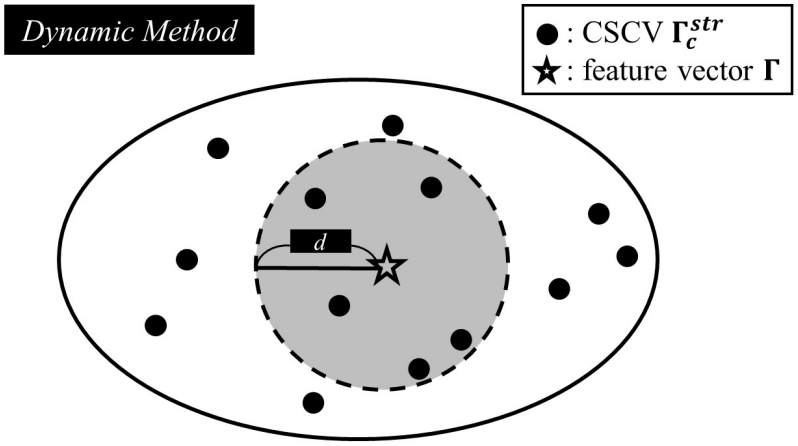


Figure 4.4: Illustration of estimation process in dynamic method.

4.3.4.1 Estimating Number of Symbols

As mentioned in Section 4.2.1, the number of "0" and "1" symbols has a different bias for each group. In the noiseless case, we can observe the bias by directly counting $\ell_{\xi,0}$ and $\ell_{\xi,1}$ and derive the feature vector $\mathbf{\Gamma}$ as given by Eq. (4.11).

In the noisy case, two estimation processes are required to derive $\ell_{\xi,0}$ and $\ell_{\xi,1}$ from a distorted codeword:

$$\hat{y}_i = y_i + e_i, \tag{4.24}$$

where e_i is AWGN with variance σ_e^2. The probability density function $pdf(\hat{y}_i)$ is given by the following equation as a Gaussian mixture model (GMM):

$$pdf(\hat{y}_i) = \pi_0 \mathcal{N}\big(\hat{y}_i; 0, \sigma_e^2\big) + \pi_1 \mathcal{N}\big(\hat{y}_i; 1, \sigma_e^2\big), \tag{4.25}$$

where $\sum_{k=0}^{1} \pi_k = 1$ and π_k represent the weights of each distribution, and

$$\mathcal{N}\big(\hat{y}_i; \mu, \sigma_e^2\big) = \frac{1}{\sqrt{2\pi\sigma_e^2}} \exp\Big(- \frac{(\hat{y}_i - \mu)^2}{2\sigma_e^2}\Big). \tag{4.26}$$

First, we estimate σ_e^2, π_0, and π_1 from all symbols $\hat{y}_i (1 \le i \le L)$ in the distorted codeword using the EM algorithm. The estimated variance is denoted by $\tilde{\sigma}_e^2$. Then, for the ξ-th group, γ_ξ is estimated from $\ell_\xi = \ell_{\xi,0} + \ell_{\xi,1}$ symbols using the EM algorithm. As the symbols $\hat{y}_i$ are distorted by noise, the probability density function in the ξ-th group is

$$\begin{aligned} pdf(\hat{y}_i)_\xi &= \frac{\ell_{\xi,0}}{\ell_\xi} \mathcal{N}\big(\hat{y}_i; 0, \tilde{\sigma}_e^2\big) + \frac{\ell_{\xi,1}}{\ell_\xi} \mathcal{N}\big(\hat{y}_i; 1, \tilde{\sigma}_e^2\big), \\ &= (1 - \gamma_\xi) \mathcal{N}\big(\hat{y}_i; 0, \tilde{\sigma}_e^2\big) + \gamma_\xi \mathcal{N}\big(\hat{y}_i; 1, \tilde{\sigma}_e^2\big). \end{aligned} \tag{4.27}$$

As the variance is estimated in the first process, the EM algorithm estimates γ_ξ in the second process. As a consequence, feature vector $\hat{\mathbf{\Gamma}}$ is calculated from the distorted codeword $\hat{\boldsymbol{y}}$.

4.3.4.2 Optimal Detection in Noisy Environment

As feature vector $\hat{\mathbf{\Gamma}}$ is distorted by noise, distance $D^{str,c}$ changes accordingly. We assume that the additive noise follows a white Gaussian distribution in the CSCV vector space. Therefore, the feature vector $\hat{\mathbf{\Gamma}}$ can be estimated using CSCVs in a noiseless environment.

However, the distortions in the pirated codeword change vector $\boldsymbol{\theta}_{\boldsymbol{c}}^{\boldsymbol{str}}$. Hence, the MAP detector must adjust vector $\boldsymbol{\theta}_{\boldsymbol{c}}^{\boldsymbol{str}}$ in accordance with the noise variance estimated from the pirated codeword $\hat{\boldsymbol{y}}$. As discussed by Meerwald and Furon [47], the adjusted parameters $\hat{\theta}_\lambda (0 \leq \lambda \leq c)$ are given by

$$\hat{\theta}_{\lambda(\hat{y}_i)} = (1-\theta_\lambda)\mathcal{N}\big(\hat{y}_i; 0, \tilde{\sigma}_e^2\big) + \theta_\lambda \mathcal{N}\big(\hat{y}_i; 1, \tilde{\sigma}_e^2\big). \tag{4.28}$$

After the above adjustment of collusion strategy $\hat{\boldsymbol{\theta}}_{\boldsymbol{c}}^{\boldsymbol{str}}$, we can execute the methods described in Section 4.3.3.

4.4 Reduction of Dimension in Estimator

In this section, we discuss the reduction of dimension in estimator.

4.4.1 Maximum Number of Colluders and CSCV

The dimension of space n_g for estimation depends on the number of maximum number c_{max} of colluders in Nuida codes as below.

$$n_g = \left\lceil \frac{c_{max}}{2} \right\rceil \tag{4.29}$$

In Section 4.3, we defined the estimation as the reduced-mapping from the vector space $\mathbb{Z}_2^L$ deriverd from the length L of codeword to a rational vector space with n_g dimension $\mathbb{R}^{n_g}$. With the increase of the number of colluders, the estimation of the number and the collusion strategy becomes difficult. In particular, as the feature vector tends to be sparse with the increase of c_{max}, it is possible further reduce the dimension of CSCV with small sacrifice of estimation accuracy.

4.4.2 Dominant Features in CSCV

Under the assumption that the vector space we measure the dominant elements in CSCV to estimate the number of colluders and collusion strategy $\boldsymbol{\theta}_{\boldsymbol{c}}^{\boldsymbol{str}}$. Then, we refer to the following four methods for the extraction of dominant elements in CSCV.

(a) The first half of elements in CSCV: $(\gamma_1^{str}, \ldots, \gamma_{ng/2}^{str})$

(b) The latter half of elements in CSCV: $(\gamma_{n_g/2}^{str}, \ldots, \gamma_{n_g}^{str})$

(c) The first and last elements in CSCV: $(\gamma_1^{str}, \gamma_{n_g}^{str})$

(d) Around half of elements centering on $n_g/2$ -th elements in CSCV: $(\gamma_{n_g/2-1}^{str}, \ldots, \gamma_{n_g/2+n_g\%2+1}^{str})$

When $\theta_\lambda^{str} = 1 - \theta_{c-\lambda}^{str}$ in collusion attack atrategy $\boldsymbol{\theta_c^{str}}$, the CSCV $\boldsymbol{\Gamma_c^{str}}$ satisfies the following condition.

$$\gamma_\xi^{str} = 1 - \gamma_{n_g-\xi}^{str} \tag{4.30}$$

On the other hand, asymmetric strategies $\theta_\lambda^{str} \neq 1 - \theta_{c-\lambda}^{str}$ such as all-0 attack and all-1 attack do not satisfy the above condition. Therefore, much information about the number of colluders and collusion strategy must be lost in case of the methods (a) and (b). In method (c), γ_1^{str} and $\gamma_{n_g}^{str}$ are respectively close to "0" and "1" because CSCVs are derived from the bias probability sequence P. As the distribution of CSCVs in the vector space is biased, misestimation for collusion strategy must be occurred. From the above reasons, we determined the methods (a), (b), and (c) were not appropriate for the reduction of elements in CSCV.

The method (d) can estimate asymmetric strategies like all-0 and all-1 attacks, and its distributions about other strategies are also sparse. Therefore, the method (d) is the most appropriate for the reduction among the above four methods. Furthermore, if the estimation with these selected features retains high accuracy, these are regarded as dominant features for the estimation. Some examples are presented in Table 4.2, where "$*$" stands for the reduced elements in CSCVs.

Table 4.2: Examples of CSCV in method (d).

c_{max}	Γ_c^{str}	method (d)
6	$(\gamma_1^{str}, \gamma_2^{str}, \gamma_3^{str})$	$(\gamma_1^{str}, \gamma_2^{str}, \gamma_3^{str})$
8	$(\gamma_1^{str}, \gamma_2^{str}, \gamma_3^{str}, \gamma_4^{str})$	$(*, \gamma_2^{str}, \gamma_3^{str}, *)$
10	$(\gamma_1^{str}, \gamma_2^{str}, \gamma_3^{str}, \gamma_4^{str}, \gamma_5^{str})$	$(*, \gamma_2^{str}, \gamma_3^{str}, \gamma_4^{str}, *)$
12	$(\gamma_1^{str}, \gamma_2^{str}, \gamma_3^{str}, \gamma_4^{str}, \gamma_5^{str}, \gamma_6^{str})$	$(*, *, \gamma_3^{str}, \gamma_4^{str}, *, *)$

4.5 Experiments for Evaluations

We conducted simulations to compare the performance of the three proposed methods.

4.5.1 Experimental Setup

The experimental setup was as follows. The number of users in a system was $N = 10^6$. The Nuida code was designed using $c_{max} = 8$, and the number of candidate values n_g for p_i was 4. The vector space of codeword $\mathbb{Z}_2^L$ as mapped to $\mathbb{R}^4$ to calculate the CSCVs. The false-positive probability was set to $\epsilon = 10^{-10}$ and $\epsilon_{FP} = (1-\epsilon)^N \approx N\epsilon = 10^{-4}$ using a rare event simulator [7, 22]. The candidate collusion strategies were $str = \{\text{maj, min, coin, all0, all1, int, WCA}\}$, and the number of colluders ranged from $\tilde{c}_{min} = 2$ to $\tilde{c}_{max} = 10$. There were 63 CSCVs ($= 7 \times 9$). The pirated codewords were produced by a collusion attack on 10^2 randomly selected combinations of c colluders.

4.5.2 Estimation Accuracy of Collusion Strategy

Assuming that the number of colluders c is known in advance, the accuracy of the estimator in the subset method can be measured. The collusion strategies are exactly the same or almost the same in some cases. When they are, the θ values are coincident. Thus, it is not necessary to distinguish such a strategy. Nevertheless, Tables 4.3, 4.4, 4.5 list the accuracy with which the collusion strategy was estimated for distances $D_1^{str,c}$ and $D_2^{str,c}$ for $2 \leq c \leq 8$. For $c = 2$, the CSCVs calculated using majority, minority, coin-flip, interleave, and WCA attack were exactly the same, and the strategies were estimated without error. The greater the number of colluders and the longer the code, the greater the accuracy. Additionally, the estimation for each code length was highly accurate. Comparing the results for $L = 1024$ and $L = 4096$, we see that these code lengths are sufficient for estimating the collusion strategy.

Table 4.3: Accuracy of estimator when c is known and $L = 1024$.

$\boldsymbol{\theta}_{\boldsymbol{c}}^{\boldsymbol{str}}$		number c of colluders						
		2	3	4	5	6	7	8
maj	$D_1^{str,c}$	100	100	100	100	100	100	100
	$D_2^{str,c}$	100	99.9	100	100	100	100	100
	$D_{Hel}^{str,c}$	100	100	100	100	100	100	100
min	$D_1^{str,c}$	-	99.0	100	100	100	100	100
	$D_2^{str,c}$	-	99.2	100	100	100	100	100
	$D_{Hel}^{str,c}$	-	99.3	100	100	100	100	100
coin	$D_1^{str,c}$	-	70.2	58.3	94.9	98.9	100	100
	$D_2^{str,c}$	-	71.1	60.4	94.9	98.9	100	100
	$D_{Hel}^{str,c}$	-	68.3	59.8	94.9	98.8	100	100
int	$D_1^{str,c}$	-	89.7	99.0	100	100	100	100
	$D_2^{str,c}$	-	87.6	98.5	99.9	100	100	100
	$D_{Hel}^{str,c}$	-	88.1	98.2	99.6	99.9	100	100
all0	$D_1^{str,c}$	100	100	100	100	100	100	100
	$D_2^{str,c}$	100	100	100	100	100	100	100
	$D_{Hel}^{str,c}$	100	100	100	100	100	100	100
all1	$D_1^{str,c}$	100	100	100	100	100	100	100
	$D_2^{str,c}$	100	100	100	100	100	100	100
	$D_{Hel}^{str,c}$	100	100	100	100	100	100	100
WCA	$D_1^{str,c}$	-	77.7	59.3	92.8	98.6	100	100
	$D_2^{str,c}$	-	78.8	60.6	93.7	99.0	100	100
	$D_{Hel}^{str,c}$	-	77.0	58.6	92.3	98.5	99.9	100
average	$D_1^{str,c}$	100	90.9	88.1	98.2	99.6	100	100
	$D_2^{str,c}$	100	90.9	88.5	98.4	99.7	100	100
	$D_{Hel}^{str,c}$	100	90.4	88.1	98.1	99.6	99.9	100

Table 4.4: Accuracy of estimator when c is known and $L = 2048$.

$\boldsymbol{\theta}_{\boldsymbol{c}}^{\boldsymbol{str}}$		number c of colluders						
		2	3	4	5	6	7	8
maj	$D_1^{str,c}$	100	100	100	100	100	100	100
	$D_2^{str,c}$	100	100	100	100	100	100	100
	$D_{Hel}^{str,c}$	100	100	100	100	100	100	100
min	$D_1^{str,c}$	-	99.8	100	100	100	100	100
	$D_2^{str,c}$	-	99.8	100	100	100	100	100
	$D_{Hel}^{str,c}$	-	99.8	100	100	100	100	100
coin	$D_1^{str,c}$	-	86.7	65.1	98.9	99.9	100	100
	$D_2^{str,c}$	-	85.2	67.5	98.9	99.9	100	100
	$D_{Hel}^{str,c}$	-	83.5	63.7	99.1	100	100	100
int	$D_1^{str,c}$	-	97.2	100	100	100	100	100
	$D_2^{str,c}$	-	96.8	99.8	100	100	100	100
	$D_{Hel}^{str,c}$	-	96.7	99.9	100	100	100	100
all0	$D_1^{str,c}$	100	100	100	100	100	100	100
	$D_2^{str,c}$	100	100	100	100	100	100	100
	$D_{Hel}^{str,c}$	100	100	100	100	100	100	100
all1	$D_1^{str,c}$	100	100	100	100	100	100	100
	$D_2^{str,c}$	100	100	100	100	100	100	100
	$D_{Hel}^{str,c}$	100	100	100	100	100	100	100
WCA	$D_1^{str,c}$	-	87.6	65.9	99.2	100	100	100
	$D_2^{str,c}$	-	88.3	67.7	99.3	100	100	100
	$D_{Hel}^{str,c}$	-	87.9	66.3	98.8	100	100	100
average	$D_1^{str,c}$	100	95.9	90.1	99.7	99.9	100	100
	$D_2^{str,c}$	100	95.7	90.7	99.7	99.9	100	100
	$D_1^{str,c}$	100	95.4	90.0	99.7	100	100	100

Table 4.5: Accuracy of estimator when c is known and $L = 4096$.

$\boldsymbol{\theta}_{\boldsymbol{c}}^{\boldsymbol{str}}$		number c of colluders						
		2	3	4	5	6	7	8
maj	$D_1^{str,c}$	100	100	100	100	100	100	100
	$D_2^{str,c}$	100	100	100	100	100	100	100
	$D_{Hel}^{str,c}$	100	100	100	100	100	100	100
min	$D_1^{str,c}$	-	100	100	100	100	100	100
	$D_2^{str,c}$	-	100	100	100	100	100	100
	$D_{Hel}^{str,c}$	-	100	100	100	100	100	100
coin	$D_1^{str,c}$	-	97.3	71.6	100	100	100	100
	$D_2^{str,c}$	-	96.9	73.6	100	100	100	100
	$D_{Hel}^{str,c}$	-	94.8	70.4	99.9	100	100	100
int	$D_1^{str,c}$	-	99.7	100	100	100	100	100
	$D_2^{str,c}$	-	99.7	100	100	100	100	100
	$D_{Hel}^{str,c}$	-	99.7	100	100	100	100	100
all0	$D_1^{str,c}$	100	100	100	100	100	100	100
	$D_2^{str,c}$	100	100	100	100	100	100	100
	$D_{Hel}^{str,c}$	100	100	100	100	100	100	100
all1	$D_1^{str,c}$	100	100	100	100	100	100	100
	$D_2^{str,c}$	100	100	100	100	100	100	100
	$D_{Hel}^{str,c}$	100	100	100	100	100	100	100
WCA	$D_1^{str,c}$	-	96.2	73.0	99.9	100	100	100
	$D_2^{str,c}$	-	96.2	72.8	99.9	100	100	100
	$D_{Hel}^{str,c}$	-	95.1	71.6	100	100	100	100
average	$D_1^{str,c}$	100	99.0	92.1	99.9	100	100	100
	$D_2^{str,c}$	100	99.0	92.3	99.9	100	100	100
	$D_1^{str,c}$	100	98.5	91.7	99.9	100	100	100

4.5.3 Determination of Radius for Dynamic Method

To use the dynamic method, it is necessary to determine radius d. As shown in Fig. 4.5, $d = 0.102$ gives the maximum traceability.

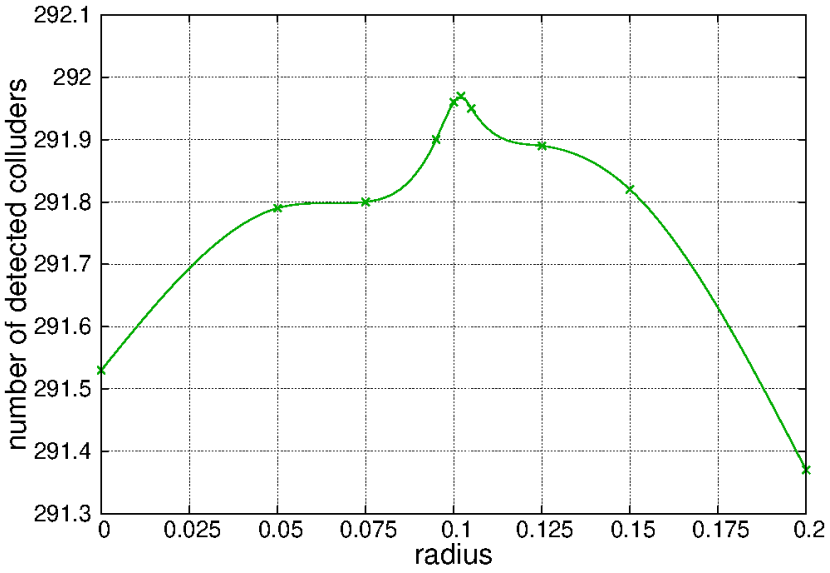


Figure 4.5: Number of detected colluders versus radius d.

The main purpose of the dynamic method is to reduce the computational cost while maintaining performance. To compare the computational cost, we measured the number n_c^{str} of CSCVs within radius $d = 0.102$ for each $\boldsymbol{\theta}_{\boldsymbol{c}}^{\boldsymbol{str}}$ and calculated its average

$$\bar{n}^{str} = \frac{1}{\tilde{c}_{max} - \tilde{c}_{min}} \sum_{c=\tilde{c}_{min}}^{\tilde{c}_{max}} n_c^{str} \tag{4.31}$$

for each strategy using the dynamic method. As the computational cost for the subset method is rational to the number $(\tilde{c}_{max} - \tilde{c}_{min})$, as explained in Section 4.3.3.2, the cost ratio R^{str} was calculated for the comparison.

$$R^{str} = \frac{\bar{n}^{str}}{\tilde{c}_{max} - \tilde{c}_{min}} \tag{4.32}$$

Table 4.6 presents the number n_c^{str} of CSCVs for $2 \leq c \leq 10$, its average $\bar{n}^{str}$, and the cost ratio R^{str}. Clearly, the dynamic method reduces the computational cost with little sacrifice in performance.

4.5.4 Traceability

Table 4.7 presents the sum of detected colluders for $2 \leq c \leq 10$, where the maximum is $54 = \sum_{c=2}^{10} c$. With the MAP detector, the collusion strategy and number of colluders

Table 4.6: Number of CSCVs within radius $d = 0.102$ measured using dynamic method and comparison of computational cost against that of subset method.

(a) $L = 1024$

$\boldsymbol{\theta}_{\boldsymbol{c}}^{\boldsymbol{str}}$	number c of colluders										
	2	3	4	5	6	7	8	9	10	$\bar{n}^{str}$	R^{str}
maj	4.93	1.55	1.60	3.70	3.76	4.79	4.86	3.41	3.47	3.56	0.40
min	4.93	2.15	1.11	1.01	1.00	1.14	1.21	1.32	1.31	1.69	0.19
coin	4.93	3.29	3.19	2.00	2.49	2.61	2.54	2.13	1.79	2.77	0.31
int	4.93	4.63	4.82	5.01	5.06	5.11	5.18	4.95	4.83	4.95	0.55
all0	1.00	1.01	1.22	1.87	2.43	3.14	3.50	3.26	2.52	2.22	0.25
all1	1.00	1.03	1.23	1.82	2.48	3.04	3.56	3.23	2.53	2.21	0.25
WCA	4.93	3.28	3.21	2.60	1.72	1.70	2.24	2.12	1.87	2.63	0.29

(b) $L = 2048$

$\boldsymbol{\theta}_{\boldsymbol{c}}^{\boldsymbol{str}}$	number c of colluders										
	2	3	4	5	6	7	8	9	10	$\bar{n}^{str}$	R^{str}
maj	5.44	1.60	1.46	3.42	3.47	5.06	5.10	3.22	3.24	3.56	0.40
min	5.44	2.86	1.05	1.00	1.02	1.19	1.26	1.52	1.55	1.88	0.21
coin	5.44	3.07	3.62	2.24	3.06	3.42	3.25	2.80	2.17	3.23	0.36
int	5.44	5.47	5.47	5.70	5.57	5.61	5.64	5.53	5.57	5.56	0.62
all0	1.00	1.00	1.17	1.78	2.48	3.11	3.49	3.25	2.51	2.20	0.24
all1	1.00	1.00	1.18	1.74	2.60	3.12	3.51	3.27	2.63	2.23	0.25
WCA	5.44	3.94	3.71	2.80	1.78	2.14	3.04	2.91	2.12	3.10	0.34

are known, so the number of detected colluders for MAP is the theoretical upper limit. Our estimator finds the closest CSCV among finite candidates, which are the well-known seven strategies and number of colluders. Hence, we also should show the performance of the detector when colluders attempt to attack using an out-of-list strategy so that misestimation occurred in our estimator. As it is difficult to check all possibilities, we checked the impact of misestimation for three collusion strategies:

- mix 1: $\boldsymbol{\theta}_{\boldsymbol{c}}^{\boldsymbol{mix1}} = (\boldsymbol{\theta}_{\boldsymbol{c}}^{\boldsymbol{int}} + \boldsymbol{\theta}_{\boldsymbol{c}}^{\boldsymbol{WCA}})/2$
- mix 2: $\boldsymbol{\theta}_{\boldsymbol{c}}^{\boldsymbol{mix2}} = (\boldsymbol{\theta}_{\boldsymbol{c}}^{\boldsymbol{int}} + \boldsymbol{\theta}_{\boldsymbol{c}}^{\boldsymbol{maj}})/2$
- mix 3: $\boldsymbol{\theta}_{\boldsymbol{c}}^{\boldsymbol{mix3}} = (\boldsymbol{\theta}_{\boldsymbol{c}}^{\boldsymbol{int}} + \boldsymbol{\theta}_{\boldsymbol{c}}^{\boldsymbol{coin}})/2$

Table 4.7 (c) shows the traceability with these strategies. The results indicate that the traceability was very close to that of the optimal detector informing the actual strategy. From the results, we can say that, if the feature vector of the pirated codeword is close to one of the CSCVs of the seven strategies, the traceability is still close to that of the optimal detector. Since our final goal is to catch as many colluders as possible, a mismatch in estimating the collusion strategy is not a problem if the traceability is very close to that of the optimal detector.

Table 4.7: Comparison of sum of detected colluders for $2 \leq c \leq 10$.

(a) $L = 1024$

	maj	min	coin	int	all0	all1	WCA	total
MAP(optimal)	21.254	53.688	9.352	9.934	30.377	30.525	8.544	163.674
Symmetric [70]	7.134	6.318	6.734	6.941	6.708	6.724	6.760	47.319
Meerwald [47]	20.419	52.813	8.828	9.308	26.185	26.152	8.012	151.717
Bias Equalizer [33]	21.130	32.639	7.669	9.694	24.499	24.617	7.642	127.890
Basic Method	21.151	53.664	9.205	9.765	30.374	30.477	8.503	163.139
Subset Method	20.919	53.634	9.201	9.924	30.238	30.965	8.502	163.383
Dynamic Method	21.178	53.669	9.283	9.878	30.329	30.722	8.508	163.567

(b) $L = 2048$

	maj	min	coin	int	all0	all1	WCA	total
MAP(optimal)	45.310	54.000	23.137	21.868	53.852	53.841	17.420	269.428
Symmetric [70]	14.667	13.313	13.871	14.332	13.878	13.920	14.062	98.043
Meerwald [47]	44.816	54.000	22.109	20.648	53.616	53.568	16.632	265.389
Bias Equalizer [33]	45.109	53.810	19.164	21.340	52.385	52.304	16.124	260.236
Basic Method	45.294	54.000	22.738	21.738	53.850	53.845	17.402	268.867
Subset Method	45.211	54.000	22.962	21.861	53.850	53.864	17.277	269.025
Dynamic Method	45.273	54.000	23.009	21.844	53.850	53.863	17.393	269.232

(c) $L = 2048$ in case of mix strategies

	mix1	mix2	mix3	total
MAP(optimal)	18.653	28.082	19.694	66.429
Symmetric [70]	14.120	14.490	14.149	42.759
Meerwald [47]	17.680	26.796	18.623	63.099
Bias Equalizer [33]	18.102	27.367	17.375	62.844
Basic Method	18.279	27.867	19.374	65.520
Subset Method	18.364	26.003	19.099	63.466
Dynamic Method	18.247	27.849	19.365	65.461

When the codewords for all users were generated, the disributor has to decice the maximum number of colluders in order to minimize the code length with keeping the

traceability as many colluders as possible. The traceabilities for the well-known seven strategies with different maximum number of colluders are presented in Table 4.8. For the mix strategies, Table 4.9 also shows the comparison of the traceabilities with mix strategies. The result indicates that our proposed methods are close to the optimal detector in any cases no matter what the distributor assumed the maximum number of colluders c_{max}

Table 4.8: Comparison of sum of detected colluders for different c_{max}.

(a) $c_{max} = 6$

	maj	min	coin	int	all0	all1	WCA	total
MAP(optimal)	47.437	54.000	22.103	22.732	53.986	53.992	18.371	272.621
Symmetric [70]	16.476	13.986	14.891	15.278	14.889	14.994	14.919	105.433
Basic Method	47.440	53.002	16.425	22.094	53.986	53.992	16.351	263.290
Subset Method	47.454	54.000	22.120	22.755	53.986	53.992	17.891	272.198
Dynamic Method	47.454	53.989	21.547	22.714	53.986	53.992	17.772	271.454

(b) $c_{max} = 10$

	maj	min	coin	int	all0	all1	WCA	total
MAP(optimal)	44.154	54.000	22.113	21.212	53.574	53.536	22.473	271.062
Symmetric [70]	14.119	12.800	13.408	13.830	13.310	13.397	13.433	94.297
Basic Method	44.133	54.000	21.570	21.087	53.573	53.539	22.219	270.121
Subset Method	44.029	54.000	21.903	21.177	53.570	53.510	22.032	270.221
Dynamic Method	44.088	54.000	21.905	21.132	53.570	53.535	22.253	270.483

(c) $c_{max} = 12$

	maj	min	coin	int	all0	all1	WCA	total
MAP(optimal)	43.652	54.000	21.532	20.837	53.305	53.227	21.209	267.762
Symmetric [70]	13.798	12.536	13.043	13.431	12.988	13.100	13.082	91.978
Basic Method	43.617	54.000	21.000	20.688	53.308	53.225	20.908	266.746
Subset Method	43.574	54.000	21.309	20.898	53.283	53.265	20.668	266.997
Dynamic Method	43.611	54.000	21.170	20.817	53.283	53.259	20.939	267.079

Table 4.9: Comparison of sum of detected colluders for different c_{max} in case of mix strategies.

(a) $c_{max} = 6$

	mix1	mix2	mix3	total
MAP(optimal)	18.937	31.145	19.714	69.796
Symmetric [70]	15.090	15.831	15.107	46.028
Basic Method	16.866	30.663	18.892	66.421
Subset Method	18.492	27.594	18.210	64.296
Dynamic Method	18.197	29.477	18.645	66.319

(b) $c_{max} = 10$

	mix1	mix2	mix3	total
MAP(optimal)	19.298	27.485	19.064	65.847
Symmetric [70]	13.592	13.960	13.591	41.143
Basic Method	18.296	27.119	18.472	63.887
Subset Method	18.581	25.168	17.798	61.547
Dynamic Method	18.323	27.096	18.367	63.786

(c) $c_{max} = 12$

	mix1	mix2	mix3	total
MAP(optimal)	18.852	26.973	18.808	64.633
Symmetric [70]	13.136	13.528	13.200	39.864
Basic Method	17.891	26.625	18.160	62.676
Subset Method	18.248	24.731	17.665	60.644
Dynamic Method	17.897	26.616	18.161	62.674

As shown in Table 4.7, and 4.8, 4.9 the results with the proposed methods exceeded the number of detectors in the optimal single detector for some cases. This is because of the probabilistic algorithm in the rare event simulator [7, 22] used to calculate the threshold. If the number of trials were increased, this would not occur. The table also shows that the traceability of the basic, subset, dynamic and reduction of dimension methods were very close to that of the optimal MAP detector for all collusion strategies. When $L = 2048$, the dynamic method outperformed the other methods. The effect of the reduction of dimension given to the traceability is small. Thus, the dominant features can be extracted by CSCV in this method.

4.5.5 Noisy Case

The total number of detected colluders for the noisy environment case are listed in Table 4.10. The signal-to-noise ratio (SNR) ranged from 0 to 10 [dB], and the number of colluders was set to 6. The values were at most 66 ($= 6 \times 11$). For the MAP detector, the collusion strategy, number of colluders, and AWGN variance were considered known. To further evaluate the accuracy of the proposed methods, we used the variance estimated by the EM algorithm in the MAP detector. Unlike the noiseless case, the total traceability of the dynamic method was better than that of the subset method in the presence of noise. The results of these experiments are illustrated in Figs. 4.6, 4.7, 4.8, 4.9, 4.10, 4.11, 4.12. These results clearly show that an optimal detector can be achieved by using the proposed estimators for the collusion attack parameters in the presence of noise.

Table 4.10: Comparison of total detected colluders in noisy case for SNR from 0–10 [dB].

(a) $L = 1024$

	maj	min	coin	int	all0	all1	WCA
MAP (known σ_e^2) [21]	20.324	58.537	2.598	2.362	42.988	43.238	1.209
MAP (estimated σ_e^2) [21]	20.288	58.566	2.603	2.356	43.012	43.218	1.179
Symmetric [70]	0.542	0.184	0.375	0.449	0.290	0.270	0.382
Meerwald [47]	18.232	53.074	2.175	1.744	32.752	33.267	0.700
Bias Equalizer [33]	19.582	36.719	1.170	2.276	33.213	33.634	0.819
Basic Method	20.119	58.422	2.338	2.277	42.907	43.136	1.203
Subset Method	20.081	58.304	2.379	2.365	42.799	43.006	1.167
Dynamic Method	20.197	58.425	2.513	2.318	42.880	43.115	1.187

(b) $L = 2048$

	maj	min	coin	int	all0	all1	WCA
MAP (known σ_e^2) [21]	58.418	65.983	29.033	28.323	64.747	64.705	19.547
MAP (estimated σ_e^2) [21]	58.470	65.980	29.044	28.399	64.767	64.719	19.602
Symmetric [70]	10.069	6.708	8.272	9.618	8.255	8.050	8.726
Meerwald [47]	57.369	65.902	26.376	24.474	63.320	63.265	17.349
Bias Equalizer [33]	58.040	64.901	18.597	27.180	63.511	63.451	14.743
Basic Method	58.296	65.983	27.353	27.914	64.729	64.688	17.821
Subset Method	58.120	65.976	28.893	28.334	64.588	64.609	17.618
Dynamic Method	58.296	65.983	28.799	28.312	64.714	64.678	17.797

(c) Total

	1024	2048
MAP (known σ_e^2) [21]	171.256	330.756
MAP (estimated σ_e^2) [21]	171.222	330.981
Symmetric [70]	2.492	59.698
Meerwald [47]	141.944	318.055
Bias Equalizer [33]	127.413	310.423
Basic Method	170.402	326.784
Subset Method	170.101	328.138
Dynamic Method	170.635	328.579

Table 4.11: $L = 2048$ in case of mix strategies.

	mix1	mix2	mix3	total
MAP (known σ_e^2) [21]	21.036	41.989	22.604	85.629
MAP (estimated σ_e^2) [21]	20.936	41.98	22.636	85.552
Symmetric [70]	9.185	10.086	9.068	28.339
Meerwald [47]	18.157	38.366	19.728	76.251
Bias Equalizer [33]	19.319	40.982	16.762	77.063
Basic Method	19.952	41.454	21.845	83.251
Subset Method	20.247	39.034	21.428	80.709
Dynamic Method	19.856	41.453	21.964	83.273

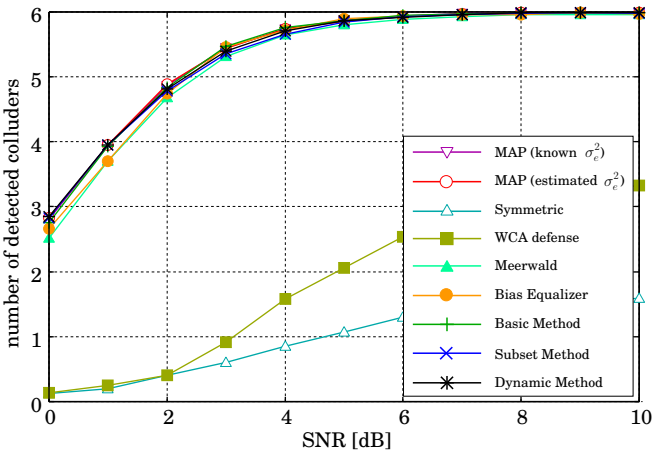


Figure 4.6: majority.

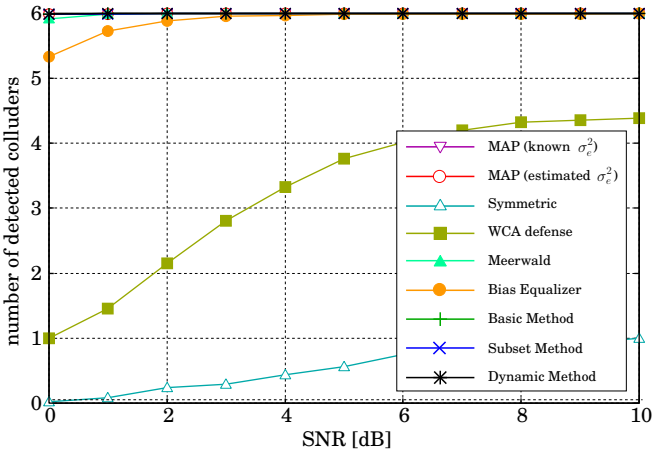


Figure 4.7: minority.

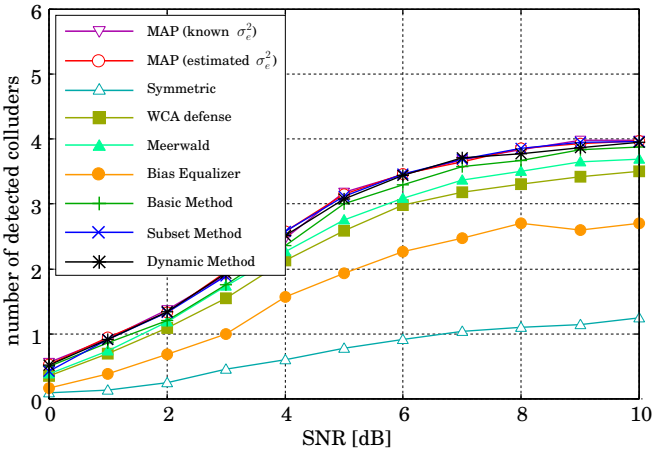


Figure 4.8: coin-flip.

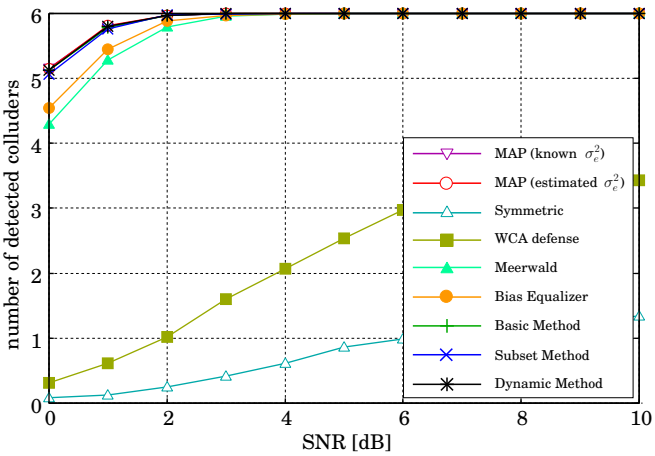


Figure 4.9: all-0.

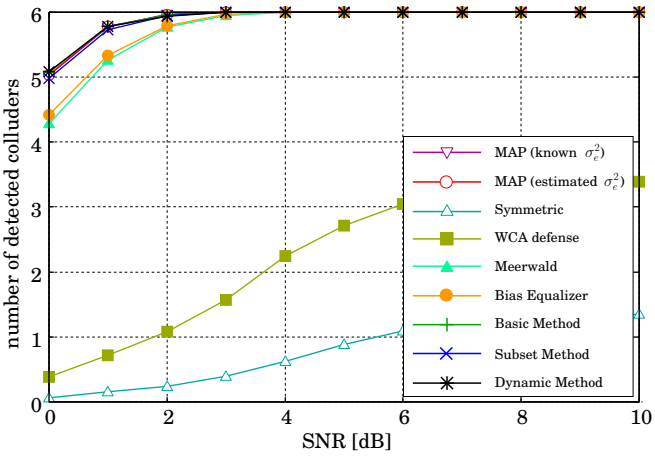


Figure 4.10: all-1.

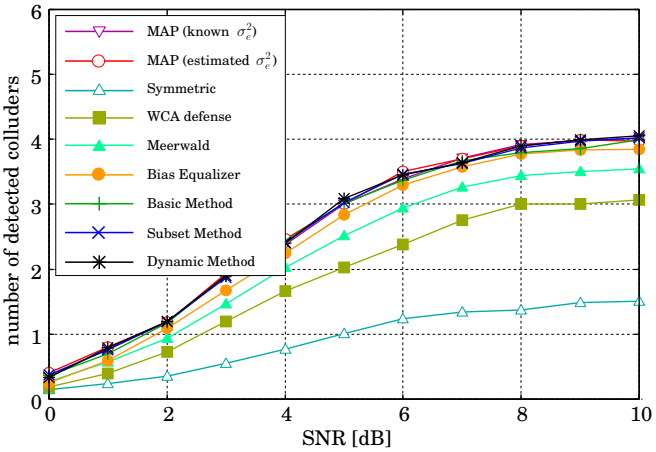


Figure 4.11: interleave.

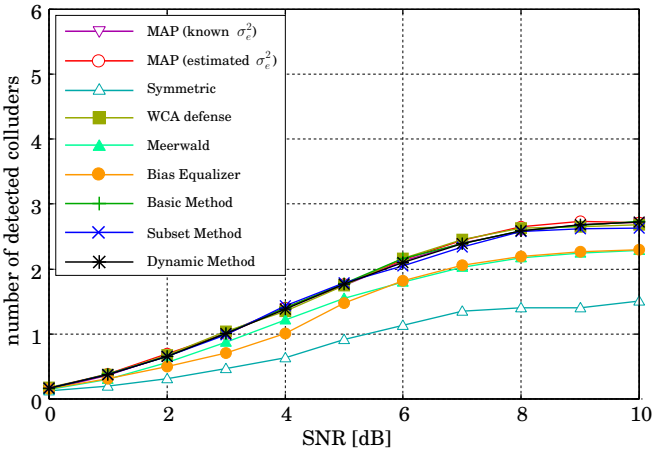


Figure 4.12: WCA.

4.5.6 Reduction of Dimension Method

The comparisons of the total number of detected colluders among the dimension reduction method and conventional methods are listed in Table 4.12. The number of colluders ranged from 2 to 10, where the maximum is 54. We applied the reduction of dimension to the Basic Method in Section 4.3.3.1, called Reduction of Dimension (RD) Method.

Table 4.12: Sum of detected colluders with Reduction of Dimension Method.

(a) $c_{max} = 8$

	maj	min	coin	int	all0	all1	WCA	total
MAP(optimal)	45.310	54.000	23.137	21.868	53.852	53.841	17.420	269.428
Basic Method	45.294	54.000	22.738	21.738	53.850	53.845	17.402	268.867
RD Method	45.307	54.000	22.789	19.731	53.849	53.841	17.171	266.688

(b) $c_{max} = 10$

	maj	min	coin	int	all0	all1	WCA	total
MAP(optimal)	44.154	54.000	22.113	21.212	53.574	53.536	22.473	271.062
Basic Method	44.133	54.000	21.570	21.087	53.573	53.539	22.219	270.121
RD Method	44.131	54.000	21.094	20.554	53.570	53.532	19.608	266.489

(c) $c_{max} = 12$

	maj	min	coin	int	all0	all1	WCA	total
MAP(optimal)	43.652	54.000	21.532	20.837	53.305	53.227	21.209	267.762
Basic Method	43.617	54.000	21.000	20.688	53.308	53.225	20.908	266.746
RD Method	43.537	54.000	20.619	17.119	53.303	53.218	18.316	260.112

In this method, we also shows the traceability with mix strategies in Table 4.13.

Table 4.13: Sum of detected colluders with Reduction of Dimension Method in mix strategies.

(a) $c_{max} = 8$

	mix1	mix2	mix3	total
MAP(optimal)	18.653	28.082	19.694	66.429
Basic Method	18.279	27.867	19.374	65.520
RD Method	18.002	27.525	19.264	64.791

(b) $c_{max} = 10$

	mix1	mix2	mix3	total
MAP(optimal)	19.298	27.485	19.064	65.847
Basic Method	18.296	27.119	18.472	63.887
RD Method	14.785	27.115	15.174	57.074

(c) $c_{max} = 12$

	mix1	mix2	mix3	total
MAP(optimal)	18.852	26.973	18.808	64.633
Basic Method	17.891	26.625	18.160	62.676
RD Method	16.959	23.370	17.754	58.083

These results in Tables 4.12, 4.13 show that it is possible to detect as many colluders as optimal detector with the least expence of accuracy.

Chapter 5

Conclusion

In this thesis, we describe DNN watermarking and pruning DNN model in Chapter 2. Furthermore, the fingerprinting codes and collusion attack model, Tardos code, Nuida code and tracking algorithm are described for the optimal detector of fingerprinting codes.

In chapter 3, We proposed a novel method to protect weight level pruning attacks in DNN watermarking by introducing the CWC. We experimentally evaluated the effect of embedding watermarks into DNN models and their robustness against pruning attacks. In addition, we evaluated the robustness of the proposed method when the DNN model is retrained after the pruning attack. We used two thresholds, T_1 and T_0, to restrict the weight parameters used to embed the watermark. Under the assumption of Gaussian or uniform distribution, T_1 can be calculated from a statistical analysis, while T_0 should be designed to consider the robustness against other possible attacks on the watermarked DNN model. However, the used CWC in our proposed study has no error-correction capability. We will consider those studies that claim CWC has error-correcting capabilities in our future work. Another future work could be designing such a model that can persist against sophisticated compressed pre-trained models.

In order to realize the optimal detector which catch as many colluders as possible against the collusion attacks known as the most threat in the fingerprinting, Chapters 4 proposed the estimator to estimate two parameters: the number of colluders and the collusion attack strategy. In the estimator, we observed the imbalance between “0” and “1” symbols in the pirated codeword and compared the distances between the feature vector observed from the pirated codeword and precalculated feature vectors. For the metrics, we also showed three methods to calculate the distances. In case that the pirated codeword is distorted in practical situations, the distribution of symbols in the pirated codeword is modeled by Gaussian Mixture Model and estimate the imbalance by Expectation – - Maximization algorithm. To suppress the computational cost, we extracted the dominant elements in feature vector and reduced the dimension in the vector space. Computer simulations revealed that the overall performance of the proposed methods was superior

to that of conventional detectors and was very close to the performance of optimal detector. As a future work, we should confirm the determination of the maximum number of colluders for generation of codewords because the dimension of estimation space and the traceability depend on how we determine the number of colluders.

Through this thesis, we have proposed two different approaches to protect multimedia content and confirmed their effectiveness. However, much remains to be evaluated for practical application. For example, in DNN watermarking, the effectiveness against layers different from the FC layer of CNNs and the effectiveness against network architectures different from CNNs need to be evaluated. In addition, attack models other than pruning need to be considered. In the case of digital fingerprinting, it is necessary to study the effectiveness when more colluders generate illegal codeword for practical use. Future interests include the study of attacks on non-binary codeword. They will be future works.

Bibliography

[1] E. Abbe and L. Zheng. Linear universal decoding for compound channels. *IEEE Trans. Inform. Theory*, 56(12):5999–6013, 2010.

[2] M. Barni, F. Bartolini, and A. Piva. Improved wavelet-based watermarking through pixel-wise masking. *IEEE Transactions on Image Processing*, 10(5):783–791, 2001.

[3] Mauro Barni and F. Bartolini. Watermarking systems engineering: Enabling digital assets security and other application. 01 2004.

[4] S. Bitan and T. Etzion. Constructions for optimal constant weight cyclically permutable codes and difference families. *IEEE Trans. Information Theory*, 41(1):77–87, 1995.

[5] D. Boneh and J. Shaw. Collusion-secure fingerprinting for digital data. *IEEE Trans. Inform. Theory*, 44(5):1897–1905, 1998.

[6] A. E. Brouwer, J. B. Shearer, N. J. A. Sloane, and W.D. Smith. A new table of constant weight codes. *IEEE Trans. Information Theory*, 36(6):1334–1380, 1990.

[7] Frédéric Cérou, Pierre Del Moral, Teddy Furon, and Arnaud Guyader. Sequential Monte Carlo for rare event estimation. *Statistics and Computing*, pages 1–14, 2011.

[8] Rafiullah Chamlawi and Asifullah Khan. Digital image authentication and recovery: Employing integer transform based information embedding and extraction. *Information Sciences*, 180(24):4909–4928, 2010.

[9] B. Chen and G.W. Wornell. Quantization index modulation: a class of provably good methods for digital watermarking and information embedding. *IEEE Transactions on Information Theory*, 47(4):1423–1443, 2001.

[10] H. Chen, B. D. Rouhani, X. Fan, O. C. Kilinc, and F. Koushanfar. Performance comparison of contemporary DNN watermarking techniques. *CoRR*, abs/1811.03713, 2018.

[11] H. Chen, B. D. Rouhani, C. Fu, J. Zhao, and F. Koushanfar. DeepMarks: A secure fingerprinting framework for digital rights management of deep learning models. In *Proc. ICMR'19*, pages 105–113, 2019.

[12] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun. *The loss surfaces of multilayer networks*. Artificial Intelligence and Statistics, 2015.

[13] I.J. Cox, J. Kilian, F.T. Leighton, and T. Shamoon. Secure spread spectrum watermarking for multimedia. *IEEE Transactions on Image Processing*, 6(12):1673–1687, 1997.

[14] Ingemar Cox, Matthew Miller, Jeffrey Bloom, Jessica Fridrich, and Ton Kalker. *Digital Watermarking and Steganography*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2 edition, 2007.

[15] Y. N. Dauphin, R. Pascanu, Ç. Gülçehre, K. Cho, S. Ganguli, and Y. Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Proc. NIPS'14*, pages 2933–2941, 2014.

[16] J. Deng, W. Dong, R. Socher, Kai Li L. Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proc. CVPR' 09*, pages 248–255, 2009.

[17] N. Denil, B. Shakibi, L. Dinh, M. A. Ranzato, and N. D. Freitas. Predicting parameters in deep learning. In *Advances In Neural Information Processing Systems*, pages 2148–2156, 2013.

[18] M. Desoubeaux, C. Herzet, W. Puech, and G. Le Guelvouit. Enhanced blind decoding of Tardos codes with new MAP-based functions. In *Proc. MMSP*, pages 283–288, 2013.

[19] X. Dong, S. Chen, and S. J. Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *NIPS'17*, pages 4860–4874, 2017.

[20] T. Etzion and A. Vardy. A new construction for constant weight codes. In *Proc. ISITA'14)*, pages 338–342, 2014.

[21] T. Furon and L. Perez-Freire. EM decoding of Tardos traitor tracing codes. In *ACM Multimedia and Security*, pages 99–106, 2009.

[22] T. Furon, L. P. Preire, A. Guyader, and F. Cérou. Estimating the minimal length of Tardos code. In *IH 2009*, volume 5806 of *LNCS*, pages 176–190. Springer, Heidelberg, 2009.

[23] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proc. PMLR'10*, volume 9, pages 249–256, 2010.

[24] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[25] Y. Guo, A. Yao, and Y. Chen. Dynamic network surgery for efficient DNNs. In *Advances In Neural Information Processing Systems*, pages 1379–1387, 2016.

[26] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, 2016.

[27] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proc. ICCV'15*, pages 1026–1034, 2015.

[28] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proc. of CVPR'16*, 2016.

[29] Yehao Kong and Jiliang Zhang. Adversarial audio: A new information hiding method and backdoor for dnn-based speech recognition models. *arXiv preprint arXiv:1904.03829*, 2019.

[30] S. K. Kumar. On weight initialization in deep neural networks. *CoRR*, abs/1704.08863, 2017.

[31] M. Kuribayashi. Tardos's fingerprinting code over AWGN channel. In *IH 2010*, volume 6387 of *LNCS*, pages 103–117. Springer, Heidelberg, 2010.

[32] M. Kuribayashi. Simplified MAP detector for binary fingerprinting code embedded by spread spectrum watermarking scheme. *IEEE Trans. Inform. Forensics and Security*, 9(4):610–623, 2014.

[33] M. Kuribayashi and N. Funabiki. Universal scoring function based on bias equalizer for bias-based fingerprinting codes. *IEICE Trans. Fundamentals*, E101-A(1):119–128, 2018.

[34] M. Kuribayashi, T. Tanaka, S. Suzuki, T. Yasui, and N. Funabiki. White-box watermarking scheme for fully-connected layers in fine-tuning model. In *IHMMsec'21*, pages 165–170, 2021.

[35] M. Kuribayashi, T. Yasui, A. Malik, and N. Funabiki. Immunization of pruning attack in DNN watermarking using constant weight code. *CoRR*, abs/2107.02961, 2021.

[36] Minoru Kuribayashi, Takuya Fukushima, and Nobuo Funabiki. Data hiding for text document in pdf file. In Jeng-Shyang Pan, Pei-Wei Tsai, Junzo Watada, and Lakhmi C. Jain, editors, *Advances in Intelligent Information Hiding and Multimedia Signal Processing*, pages 390–398, Cham, 2018. Springer International Publishing.

[37] Minoru KURIBAYASHI, Takuya FUKUSHIMA, and Nobuo FUNABIKI. Robust and secure data hiding for pdf text document. *IEICE Transactions on Information and Systems*, E102.D(1):41–47, 01 2019.

[38] Minoru Kuribayashi, Takuro Tanaka, and Nobuo Funabiki. Deepwatermark: Embedding watermark into dnn model. In *2020 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, pages 1340–1346, 2020.

[39] T Laarhoven. Capacities and capacity-achieving decoders for various fingerprinting games. In *Proc. IH&MMSec2014*, pages 123–134, 2014.

[40] Erwan Le Merrer, Patrick Perez, and Gilles Trédan. Adversarial frontier stitching for remote neural network watermarking. *Neural Computing and Applications*, 32(13):9233–9244, 2020.

[41] Y. Li, B. Tondi, and M. Barni. Spread-transform dither modulation watermarking of deep neural network. *Journal of Information Security and Applications*, 63:103004, 2021.

[42] Yue Li, Hongxia Wang, and Mauro Barni. A survey of deep neural network watermarking techniques. *Neurocomputing*, 461:171–193, 2021.

[43] S.D. Lin, Shih-Chieh Shie, and Han Yi Guo. Improving the robustness of dct-based image watermarking against jpeg compression. In *2005 Digest of Technical Papers. International Conference on Consumer Electronics, 2005. ICCE.*, pages 343–344, 2005.

[44] Y. T. Lin, J. L. Wu, and C. H. Huang. Concatenated construction of traceability codes for multimedia fingerprinting. *Optical Engineering*, 46(10):107202.1–107202.15, 2007.

[45] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Fine-pruning: Defending against backdooring attacks on deep neural networks. In Michael Bailey, Thorsten Holz, Manolis Stamatogiannakis, and Sotiris Ioannidis, editors, *Research in Attacks, Intrusions, and Defenses*, pages 273–294, Cham, 2018. Springer International Publishing.

[46] F. J. MacWilliams and N. J. A. Sloane. *The theory of error-correcting codes*. Amsterdam: North-Halland, 1977.

[47] P. Meerwald and T. Furon. Towards practical joint decoding of binary Tardos fingerprinting codes. *IEEE Trans. Inform. Forensics and Security*, 7(4):1168–1180, 2012.

[48] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz. Pruning convolutional neural networks for resource efficient transfer learning. *CoRR*, abs/1611.06440, 2016.

[49] P. Moulin. Universal fingerprinting: Capacity and random-coding exponents. *Proc. ISIT 2008*, pages 220–224, 2008.

[50] Y. Nagai, Y. Uchida, S. Sakazawa, and S. Satoh. Digital watermarking for deep neural networks. *International Journal of Multimedia Information Retrieval*, 7:3–16, 2018.

[51] Q. A. Nguyen, L. Gyorfi, and J. L. Massey. Constructions of binary constant-weight cyclic codes and cyclically permutable codes. *IEEE Trans. Information Theory*, 38(3):940–949, 1992.

[52] N. Nikolaidis and I. Pitas. Robust image watermarking in the spatial domain. *Signal Processing*, 66(3):385–403, 1998.

[53] K. Nuida, S. Fujitsu, M. Hagiwara, T. Kitagawa, H. Watanabe, K. Ogawa, and H. Imai. An improvement of discrete Tardos fingerprinting codes. *Designs, Codes and Cryptography*, 52(3):339–362, 2009.

[54] K. Nuida, M. Hagiwara, H. Watanabe, and H. Imai. Optimization of Tardos's fingerprinting codes in a viewpoint of memory amount. In *Proc. IH 2007*, volume 4567 of *LNCS*, pages 279–293. Springer, Heidelberg, 2008.

[55] J. J. Oosterwijk, B. Škorić, and J. Doumen. A capacity-achieving simple decoder for bias-based traitor tracing schemes. *IEEE Trans. Inform. Theory*, 61(7):3882–3900, 2015.

[56] C.I. Podilchuk and E.J. Delp. Digital watermarking: algorithms and applications. *IEEE Signal Processing Magazine*, 18(4):33–46, 2001.

[57] V.M. Potdar, S. Han, and E. Chang. A survey of digital image watermarking techniques. In *INDIN '05. 2005 3rd IEEE International Conference on Industrial Informatics, 2005.*, pages 709–716, 2005.

[58] B. D. Rouhani, H. Chen, and F. Koushanfar. DeepSigns: An end-to-end watermarking framework for ownership protection of deep neural networks. In *Proc. ASPLOS'19*, pages 485–497, 2019.

[59] J. P. M. Schalkwijk. An algorithm for source coding. *IEEE Trans. Information Theory*, IT-18(3):395–399, 1972.

[60] A. Simone and B. Škorić. Accusation probabilities in Tardos codes: beyond the gaussian approximation. *Designs, Codes and Cryptography*, 63(3):379–412, 2012.

[61] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proc. of ICLR'15*, 2015.

[62] Amit Kumar Singh, Nomit Sharma, Mayank Dave, and Anand Mohan. A novel technique for digital image watermarking in spatial domain. In *2012 2nd IEEE International Conference on Parallel, Distributed and Grid Computing*, pages 497–501, 2012.

[63] D. H. Smith, L. A. Hughes, and S. Perkins. A new table of constant weight codes of length greater than 28. *The Electron Journal of Combination*, 13, 2006.

[64] J. N. Staddon, D. R. Stinson, and R. Wei. Combinatorial properties of frameproof and traceability codes. *IEEE Trans. Inform. Theory*, 47(3):1042–1049, 2001.

[65] G. Tardos. Optimal probabilistic fingerprint codes. In *Proc. STOC 2003*, pages 116–225, 2003.

[66] W. Trappe, M. Wu, Z. J. Wang, and K. J. R. Liu. Anti-collusion fingerprinting for multimedia. *IEEE Trans. Signal Process.*, 51(4):1069–1087, 2003.

[67] Tsz Kin Tsui, Xiao-Ping Zhang, and Dimitrios Androutsos. Color image watermarking using multidimensional fourier transforms. *IEEE Transactions on Information Forensics and Security*, 3(1):16–28, 2008.

[68] Y. Uchida, Y. Nagai, S. Sakazawa, and S. Satoh. Embedding watermarks into deep neural networks. In *Proc. ICMR'17*, pages 269–277, 2017.

[69] Kodathala Sai Varun, Ajay Kumar Mandava, and Rakesh Chowdary. Robust dwt-svd domain image watermarking based on iterative blending. *Journal of Physics: Conference Series*, 2070(1):012111, nov 2021.

[70] B. Škorić, S. Katzenbeisser, and M. Celik. Symmetric Tardos fingerprinting codes for arbitrary alphabet sizes. *Designs, Codes and Cryptography*, 46(2):137–166, 2008.

[71] J. Wang, H. Hu, X. Zhang, and Y. Yao. Watermarking in deep neural networks via error back-propagation. In *IS&T Electronic Imaging, Media Watermarking, Security and Forensics*, 2020.

[72] T. Wang and F. Kerschbaum. Attacks on digital watermarks for deep neural networks. In *Proc. ICASSP'19*, pages 2622–2626, 2019.

[73] Yumin Wang and Hanzhou Wu. Protecting the intellectual property of speaker recognition model by black-box watermarking in the frequency domain. *Symmetry*, 14(3):619, 2022.

[74] Hanzhou Wu, Gen Liu, Yuwei Yao, and Xinpeng Zhang. Watermarking neural networks with watermarked images. *IEEE Transactions on Circuits and Systems for Video Technology*, 31(7):2591–2601, 2021.

[75] M. Wu, W. Trappe, Z. J. Wang, and K. J. R. Liu. Collusion resistant fingerprinting for multimedia. *IEEE Signal Processing Magazine*, 21(2):15–27, 2004.

[76] Y. Yacobi. Improved Boneh-Shaw content fingerprinting. In *Proc. CT-RSA 2001*, volume 2020 of *LNCS*, pages 378–391. Springer-Verlag, 2001.

[77] X. Zhao, Y. Yao, H. Wu, and X Zhang. Structural watermarking to deep neural networks via network channel pruning. *CoRR*, abs/2107.08688, 2021.