UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Contribution to the construction of fingerprinting and watermarking schemes to protect mobile agents and multimedia content

by

Joan Tomàs-Buliart

Director:

Prof. Marcel Fernández-Muñoz

A dissertation submitted to the **Department of Network Engineering** and the committee on graduate studies of **Universitat Politècnica de Catalunya** in partial fulfilment of the requirements for the degree of doctor

Barcelona, September 2018

This document has been produced using LaTeX2ε.

To Silvia, Joan and Laia,
my mother,
my two fathers
and my family

"Tornarem a sofrir,
tornarem a lluitar
i tornarem a vèncer"
Lluís Companys i Jover

"It is only in the mysterious equations of love
that any logic or reasons can be found"
John Nash (A beautiful mind)

AGRAÏMENTS

Aquesta tesi ha estat molt llarga, és cert, per aquest motiu en especial i per molts d'altres vull agrair profundament a totes les persones que m'han ajudat i animat en el camí. Especialment:

- Al professor Marcel Fernández i Muñoz, per la direcció, el suport, la paciència, la insistència, la crítica constructiva i l'orientació. De ben segur que sense ell aquesta tesi no s'hauria acabat.
- Al professor Miquel Soriano i Ibáñez, per haver cregut que podia aportar quelcom a la recerca i recolzar-me en el camí, pels consells, tant els acadèmics, pels que li estic profundament agraït, però sobretot pels personals.
- Al professor Emilio Sanvicente i Gargallo, per fer que els turbo-codis semblessin senzills, la descodificació lògica i la versemblança evident.
- Als companys del Information Security Group, per l'ajuda, la col·laboració, les idees i el bon humor. Heu estat uns magnífics companys de viatge.
- Als meus projectistes, el Joan, el Sergi, el Christian, el Ricardo, el Federico, el Xavier i, especialment, el Roger i l'Ana, sense vosaltres aquesta tesi no hauria estat possible.
- Als membres del Departament d'Enginyeria Telemàtica de la UPC, especial a la línia de SERTEL, per haver-me ajudat en tot el que us he anat demanant durant aquest 14 anys.
- A la meva família i amics que sempre han cregut que el dia d'avui arribaria.

Research of this work was supported in part by the Spanish Government through projects TSI2005-07293-C02-01 (SECONNET), TEC2008-06663-C03-01 (P2PSEC), and TEC2011-26491 “COPPI”, by the Spanish Ministry of Science and Education with CONSOLIDER CSD2007-00004 (ARES), by Generalitat de Catalunya with the grants 2005 SGR 01015 and 2009 SGR 01362 to consolidated research groups and by Secretaría de Estado de Universidades e Investigación of Spanish Government through the grant BES-2006-13976.

ACRONYMS

AG Algebraic Geometric Code

API Application Programming Interface

BCH Bose–Chaudhuri–Hocquenghem codes

BER Bit Error Ratio

CDMA Code-Division Multiple-Access

CIA Cooperative Itinerant Agent

CRL Certificate Revocation List

CSCFI Collusion Secure Convolutional Fingerprinting Information

DCT Discrete Cosine Transform

DRM Digital Right Management

FBF Fingerprinting Branch Function

FFT Fast Fourier Transform

FP Frameproof

HoRA Host Revocation Authority

HRL Host Revocation List

HVS Human Visual System

IDS Intrusion Detection Systems

IPP Identifiable Parent Property

LLR Log-Likelihood Ratio

MAP Maximum A-posteriori Probability Algorithm

MAW Mobile Agent Watermarking

MFBF Modified Fingerprinting Branch Function

MFD Maximum Free Distance

MFDLRC Maximum Free Distance Low-Rate Convolutional Codes

ML Maximum-likelihood

ODS Optimum Distance Spectrum

PKI Public Key infrastructure

PSNR Peak Signal-to-Noise Ratio

QIM Quantization Index Modulation

RSC Recursive Systematic Convolutional

SDS Superior Distance Spectrum

SFP Secure Frameproof

SNR Signal-to-Noise Ratio

SOVA Soft Output Viterbi Algorithm

SVBBSW Self-Validating Branch-Based Software Watermarking

TA Traceability Property

TFC Turbo Fingerprinting Codes

TTP Third Trusted Party

WNR Watermark-to-Noise Ratio

ABSTRACT

The main characteristic of fingerprinting codes is the need of high error-correction capacity due to the fact that they are designed to avoid collusion attacks which will damage many symbols from the codewords. Moreover, the use of fingerprinting schemes depends on the watermarking system that is used to embed the codeword into the content and how it honors the marking assumption. In this sense, even though fingerprinting codes were mainly used to protect multimedia content, using them on software protection systems seems an option to be considered.

This thesis, studies how to use codes which have iterative-decoding algorithms, mainly turbo-codes, to solve the fingerprinting problem. Initially, it studies the effectiveness of current approaches based on concatenating tradicioanal fingerprinting schemes with convolutional codes and turbo-codes. It is shown that these kind of constructions ends up generating a high number of false positives. Even though this thesis contains some proposals to improve these schemes, the direct use of turbo-codes without using any concatenation with a fingerprinting code as inner code has also been considered. It is shown that the performance of turbo-codes using the appropiate constituent codes is a valid alternative for environments with hundreds of users and 2 or 3 traitors. As constituent codes, we have chosen low-rate convolutional codes with maximum free distance.

As for how to use fingerprinting codes with watermarking schemes, we have studied the option of using watermarking systems based on informed coding and informed embedding. It has been discovered that, due to different encodings available for the same symbol, its applicability to embed fingerprints is very limited. On this sense, some modifications to these systems have been proposed in order to properly adapt them to fingerprinting applications. Moreover the behavior and impact over a video produced as a collusion of 2 users by the YouTube's service has been studied. We have also studied the optimal parameters for viable tracking of users who have used YouTube and conspired to redistribute copies generated by a collusion attack.

Finally, we have studied how to implement fingerprinting schemes and software watermarking to fix the problem of malicious hosts on mobile agents platforms. In this regard, four different alternatives have been proposed to protect the agent depending on whether you want only detect the attack or avoid it in real time. Two of these proposals are focused on the protection of intrusion detection systems based on mobile agents. Moreover, each of these solutions has several implications in terms of infrastructure and complexity.

CONTENTS

iii

Agraïments **v**

Acronyms **ix**

Abstract **xi**

I Overview **1**

1 Introduction **3**

1.1 About this thesis 3

1.2 Motivation and objectives 4

1.2.1 Objectives on fingerprinting codes and schemes 4

1.2.2 Objectives on secure e-commerce of multimedia content 5

1.2.3 Objectives on mobile agents protection 6

1.3 Main contributions of this thesis 7

1.3.1 Contributions related to fingerprinting codes and schemes . . . 7

1.3.2 Contributions related to secure e-commerce of multimedia content 7

1.3.3 Contributions related to mobile agents protection 8

2 State of the art **9**

2.1 Digital Watermarking 10

2.1.1 Differences between watermarking and cryptography 10

2.1.2 Classical image watermarking systems 12

2.1.3 Common concepts in actual watermarking schemes 14

2.1.4 Spread Spectrum Modulation 16

2.2 Digital Fingerprinting 18

2.2.1 Properties of fingerprinting codes 19

2.2.2 Types of fingerprinting codes 20

2.2.3 Comparison of the most significant existing fingerprinting schemes . . . 27
2.3 Codes with iterative decoding . . . 29
2.3.1 Turbo Codes . . . 31
2.4 Software Watermarking . . . 33
2.4.1 Classification of Software Watermarks . . . 34
2.4.2 Threat Model for Software Copyright Protection . . . 34
2.4.3 Dynamic Graph Watermarking . . . 35
2.4.4 Self-Validating Branch-Based Software Watermarking by Myles *et* al. . . . 37
2.5 Mobile agents . . . 40

II Contributions related to fingerprinting codes and schemes 45

3 Improvements of existent convolutional-like fingerprinting codes 47
3.1 Introduction . . . 47
3.2 Definition . . . 49
3.3 Boneh-Shaw fingerprinting model . . . 50
3.3.1 n-secure codes . . . 51
3.3.2 Logarithmic Length c-Secure Codes . . . 52
3.4 Impr. of Collusion Secure Convolutional Fingerprinting Information Codes . . . 52
3.4.1 Collusion Secure Convolutional Fingerprinting Information Codes . . . 53
3.4.2 A new critical performance analysis . . . 55
3.4.3 Guidelines for minimizing the effect of false positives . . . 59
3.5 New considerations about the correct design of Turbo Fingerprinting Codes . . . 60
3.5.1 Turbo Codes . . . 60
3.5.2 Turbo Fingerprinting Scheme . . . 62
3.5.3 A new critical performance analysis . . . 64
3.5.4 Proposed improvements and open problems . . . 66
3.6 Conclusions . . . 68

4 Use of Turbo Codes with Low-Rate Convolutional Constituent Codes 73
4.1 Introduction . . . 73
4.1.1 The novel contribution . . . 74
4.2 Definitions and previous results . . . 75
4.2.1 Turbo Codes . . . 76

4.2.2 Maximum free distance low-rate convolutional codes 76
4.2.3 Traceability Codes 77
4.3 Family of turbo fingerprinting codes for coalitions of size two 78
4.3.1 Code construction 78
4.3.2 Family construction 79
4.4 Security analysis 79
4.4.1 Study about the performance of the presented codes depending on constituent codes and the number of supported users 81
4.4.2 Length comparison with other well-known fingerprinting constructions 83
4.4.3 Puncturing effects on proposed codes 85
4.4.4 On the selected algorithm and implementation details of the watermarking layer 86
4.4.5 Innocent-user framing probability versus Watermarking-to-Noise Ratio 89
4.4.6 Effect of the use of a repetition code in the performance of the whole system 89
4.5 Conclusions 92

III Contributions related to secure e-commerce of multimedia content 93

5 Traitor tracing over YouTube video service - Proof of concept 95
5.1 Introduction 95
5.2 Scenario description 97
5.3 Watermarking Layer 98
5.3.1 Watermarking in the frequency domain 99
5.3.2 Secure Spread Spectrum 100
5.4 Fingerprinting Layer 101
5.4.1 Background on coding theory 102
5.4.2 Construction of a Concatenated Fingerprinting Code 103
5.4.3 Overview of the Fingerprinting Concatenated Decoding Algorithm 103
5.5 YouTube Broadcast video service 104
5.5.1 Technical notes 105
5.6 Our implementation 105
5.6.1 Sequence generator 108
5.6.2 Collusion attack generator 110
5.6.3 Traitor Tracing 110
5.6.4 External tools 110

5.7 Results . . . 111
5.7.1 How to choose the correct α. . . . 112
5.7.2 Traitors retrieval performance after collusion attacks. . . . 114
5.8 Conclusions . . . 115

6 Development of a platform for the copyright protection 117
6.1 Introduction . . . 117
6.2 Working Scenario . . . 118
6.3 Implementation Details . . . 120
6.3.1 Watermarking Layer . . . 120
6.3.2 Fingerprinting Layer . . . 120
6.3.3 Implementation Details of Digital Rights Protection . . . 121
6.4 Entities and Collaboration . . . 124
6.4.1 Platform Functionalities . . . 126
6.4.2 Stock Management . . . 128
6.4.3 System Architecture . . . 129
6.4.4 Platform User Interface . . . 132
6.5 Conclusion . . . 134

IV Contributions related to Mobile Agent Protection 137

7 Execution integrity of mobile agents in intrussion detection systems 139
7.1 Introduction . . . 140
7.2 Background . . . 140
7.2.1 Software watermarking and fingerprinting . . . 140
7.2.2 Intrusion Detection Systems . . . 141
7.2.3 IDS based on autonomous agents . . . 141
7.2.4 Risks in an IDS based on agents . . . 142
7.3 Mobile Agent integrity System . . . 143
7.3.1 Scheme proposal . . . 143
7.3.2 Discussion . . . 147
7.4 Improvement of Cooperative Itinerant Agents platform . . . 147
7.4.1 Protecting agents against replay attacks . . . 149
7.4.2 Using a matrix of marks . . . 149
7.4.3 Code obfuscation . . . 154
7.4.4 Mark embedding . . . 155
7.4.5 Discussion . . . 156
7.5 Conclusions . . . 156

8 Protection of MA execution using an external sentinel 157
8.1 Introduction . . . 157
8.2 General Concepts . . . 159
8.3 Self-Validating Branch-Based Software Watermarking with external sentinel . . . 159
8.4 Security analysis . . . 161
8.5 Implementation aspects . . . 163
8.6 Conclusions . . . 164

9 An infrastructure for detecting and punishing malicious hosts 165
9.1 Introduction . . . 165
9.2 Background . . . 167
9.2.1 Malicious Hosts . . . 167
9.2.2 Software Watermarking . . . 168
9.3 Mobile Agent Watermarking (MAW) . . . 168
9.3.1 Watermark Embedding . . . 169
9.3.2 Watermark Transference . . . 170
9.3.3 Detecting Manipulations . . . 171
9.3.4 Advantages and Drawbacks of MAW . . . 172
9.3.5 Design of the Watermarks for MAW . . . 173
9.3.6 Implementation of MAW using the CT Algorithm . . . 175
9.4 Punishing Attacks with the HoRA . . . 179
9.4.1 Status Checking . . . 179
9.4.2 Host Revocation . . . 180
9.4.3 Summarizing the Overall Process . . . 181
9.5 Conclusions . . . 182

V Final remarks 183

10 Conclusions and future work 185
10.1 Conclusions . . . 185
10.2 Future research work . . . 188

Own References 191
JCR . . . 191
LNCS . . . 191
International conferences . . . 192
Spanish conferences . . . 192

References 195

LIST OF FIGURES

2.1 Differences between watermarking and cryptography 11
2.2 Generic embedding of marks 15
2.3 Generic mark extraction 15
2.4 DCT coefficients 16
2.5 Dual Turbo Encoder/Decoder with ratio $r = \frac{1}{3}$. 32
2.6 Scheme of Dynamic Graph Watermarking 36
2.7 Example of embedding n=4453 by means of Radix-k 37
2.8 Some instructions (jump or call for instance) are converted to calls to a branch function. The next step in the program execution will be managed by this branch function. 39
2.9 Tamper detection mechanism implemented with checksums and branch functions. 39
2.10 Mobile Agent Scenario 40

3.1 State diagram for (2,1,2) Convolutional code 56
3.2 Paths in a trellis diagram corresponding to two colluders that can create false positives 57
3.3 False Positive Probability *vs.* Number of users 59
3.4 False Positive Probability *vs.* Memory of Convolutional Code 59
3.5 Dual Turbo Encoder/Decoder with ratio $r = \frac{1}{3}$. 62
3.6 Turbo fingerprinting scheme for 2 traitors. 63
3.7 Bit probability error of a TFC with generator sequences constituent RSC $(53,75)_8$ over collusion attack decoded using likelihood information. 67
3.8 Bit probability error of a TFC with generator sequences constituent RSC $(53,75)_8$ concatenated with several error correcting codes over collusion attack decoded using likelihood information. 69
3.9 % of detecting 0, 1 or 2 traitors after a collusion attack of 2 traitors by the use of TFC with correlation decoding. 70

4.1 Feed-forward concolutional encoder for a rate $R = 1/m$ and constraint length $k = 7$. 76

4.2 Schema of the simulated scenario. 81
4.3 Evolution of error probability versus attack noise level (in dB) and internal convolutional code ratio R for a user group of 2^{25} and $c = 3$. 84
4.4 Effect of puncturing the parity bits generated by the RSC encoders of a turbo encoder. The turbo fingerprinting code has two identical RSC encoders with generator polynomials F_{s_1}, F_{s_2}, F_{s_3}, F_{s_4}, F_{s_5}. Simulation results obtained using a turbo code with $c = 2$, $SNR = 10dB$ and constraint length $K = 7$. 86
4.5 Evolution of error probability versus attack noise level (in dB), for a user group of 2^8, using a turbo fingerprinting code with $c = 2$, constraint length $K = 7$ and internal convolutional code ratio $R = \frac{1}{15}$. 90
4.6 Evolution of error probability versus the number r of repetitions of the codeword of the users (r is indicated near every point in the figure), for a user group of 2^8, using different turbo fingerprinting codes with $c = 2$, constraint length $K = 7$, internal convolutional code ratios $R = \{\frac{1}{5}, \frac{1}{10}, \frac{1}{15}, \frac{1}{20}, \frac{1}{25}\}$ and WNR=−3.5218 dB. 91
4.7 Evolution of error probability versus the number of repetitions of the codeword of the users (r), for a user group of 2^8, using a turbo fingerprinting code with $c = 2$, constraint length $K = 7$, internal convolutional code ratio $R = \frac{1}{15}$ and WNR=−3.5218 dB. 92

5.1 Overall system 98
5.2 Watermarking process schema 101
5.3 Collusion attack process. 106
5.4 Traitors tracing workflow. 107
5.5 Tools developed to perform our proof of concept. 109
5.6 Corpus of videos used in our simulations. These images have Copyright (1996) David Sarnoff Research Center, Inc. and are availabel at [108]. All of them are MPEG-2 elementary streams with a resolution of 352×288 pixels and a bit rate of 1.5 Mbps. 112
5.7 Watermarking layer performance. 113
5.8 Results of traitor tracing in scenarios with and without collusion and different α values. 114

6.1 Libavfilter working flow 122
6.2 Graphical representation of platform main functionalities. 126
6.3 Diagram of the generic platform modules 129
6.4 Deployment diagram of the services proposed in the platform 131
6.5 System welcome page and product detailed information view 132
6.6 Products list and available menu options (right side) 133

7.1 MAIS System Architecture 145
7.2 Transceivers verification by cooperative agents. 148

8.1 Schema of self-validating branch-based software watermarking with external control operation. 161
8.2 Control of critically distributed infrastructures with mobile agents and different `sentinels`. 162

9.1 Container generation process. 176
9.2 Working of MAW proposal 181

LIST OF TABLES

2.1 Feasible Set according to the different definitions 24
2.2 Characteristics of some fingerprinting codes. 27

4.1 Simulation results obtained using a turbo fingerprinting code with $c = 2$, $SNR = 10dB$, constraint length $K = 7$. The turbo fingerprinting code has two identical convolutional codes with generator polynomials F_{s_1}, F_{s_2}, F_{s_3}, F_{s_4}, F_{s_5}. 83
4.2 Length comparison with Boneh-Shaw and Tardos contructions considering a collusion of 2 attackers (c=2). 85
4.3 Length comparison with Boneh-Shaw and Tardos contructions considering a collusion of 3 attackers (c=3). *Note that in BS and Tardos constructions, the channel is considered noiseless, nevertheless, some noise is considered in the proposed construction (not considered in BS or Tardos).* 85

5.1 PSNR between original sequences and sequences after YouTube process. . 113

6.1 Tests results for movie sample 124

7.1 Matrix of marks 151
7.2 Fixed values used in the example. 151
7.3 Random values issued every time that the CIA Matrix of marks 152
7.4 Matrix Positions *versus* set of possible sub marks to verify. 153

Part I

Overview

CHAPTER 1

INTRODUCTION

1.1 About this thesis

This thesis deals with the construction of fingerprinting schemes taking into account the problems in their actual implementations, whether they are used for the protection of multimedia content, such as software in the form of mobile agents. Specifically, the areas of study in which the contributions of this thesis are framed could be summarized in:

- **Watermarking techniques**, that is, how a mark, which contains a certain information, can be embedded within a certain digital content (mainly videos and software parts) in order to guarantee a set of security properties.

- The way these marks have to be generated to protect the contents against confabulation attacks is treated with **fingerprinting schemes**. A collusion attack occurs when several users who have access to different copies of the same content decide to put them in common in order to identify parts of the embedded marks and produce a new copy that contains parts of the marks of the attackers. Once this copy has been obtained, it is illegally distributed.

- A typology of codes commonly used in fingerprinting schemes is that ones with iterative decoding. Specifically, the suitability of the **convolutional codes** and their combination in the form of **turbo-codes** has been studied to be used in fingerprinting schemes.

This thesis is structured within three main parts. Each one related to one of these areas. Moreover, the next chapter describes the state-of-the-art on these areas. Finally, in the last chapter, all the conclusions and open research lines are summarized.

1.2 Motivation and objectives

In the previous section we have presented a brief introduction of the three areas in which this thesis has made contributions and how they are related to each other.

Basically, in the part II the fingerprinting codes based on iterative decoding codes are studied. The way to integrate fingerprinting codes with watermarking schemes and how they react in a hostile environment like YouTube is addressed in the part III. Finally, in the part IV, we push watermaking and fingerprinting one step further by applying its principles to protect the execution of mobile agents.

The details about of the different chapters that make up each part are summarized in more detail below.

1.2.1 Objectives on fingerprinting codes and schemes

First we analyse and aim to reduce the impact of the problem of false positives detected in the scheme presented in *Convolutional Fingerprinting Information Codes* by Zhu *et al.* [122]. These codes are the result of concatenating convolutional codes with the Boneh-Shaw [11] fingerprinting scheme. These codes have a design deficiency which provokes that they will not be necessarily c-secured with ϵ-error. This failure is due to the fact that the Viterbi standard error probability analysis can not directly be applied to the detection algorithm proposed by the authors. So, the main objectives in this case are:

- to show the problem of false positives that this construction has, that is to say, to prove how an authorized user can be accused of illegal redistribution
- to quantify, by means of a threshold, this probability of false positive and justify it both analytically and by means of simulation
- to sketch some master lines for the correct design of this family's codes.

A similar approach was followed by Zhang *et al.* in [121]. Basically, these new codes are formed by the composition of an outer turbo-code with an inner code based on the Boneh-Shaw scheme [11]. This proposal has some problems due to the symbol-by-symbol collusion attack performed by pirates is not treated efficiently by the decoding algorithm. The objectives are:

- present the problems detected in order to facilitate the design of new schemes of turbo fingerprinting codes that allow the correct identification of, at least, one attacker,
- improve the codes using the likelihood information of the non-detected coefficients by the attackers, that is, conditioning the turbo-decoder depending on whether the internal code has detected that a symbol has not been modified as a result of confabulation,
- increase the performance of codes by using the correlation between the words of the code and the likelihood obtained from the decoding of the collapsed word with the drawback of increasing the computational cost of decoding.

Finally, we explore the use of turbo-codes in fingerprinting schemes. Specifically, it is proposed to use as constituent codes those presented by Frenger *et al.* in [42] taking advantage of the fact that they are convolucional codes of low ratio with maximum free distance. Specifically to:

- introduce a new family of turbo-codes that are safe against collusion attacks of 2 traitors,
- show how efficiently it can be the tracking of attackers using turbo-decoding algorithms,
- compare the length of this new code family with the codes of Boneh-Shaw [11] and Tardos [107]

1.2.2 Objectives on secure e-commerce of multimedia content

In this part, we present two different developments. First, a development that has been done in order to be able to identify the users that have worked together to generate an illegal copy of a content and subsequently they have been distributed through the YouTube service. The objectives of this development are:

- to develop the necessary software to embed fingerprinting codewords using a watermarking algorithm (specifically one variant of the well-known algorithm presented by Cox to [27]) in videos in MPEG-2 format,
- to perform collusions of 2 traitors and distribute them through the YouTube service,
- to evaluate the impact of the video processing carried out by the YouTube service on the marked videos when it comes to tracking the users that have participated in the match.

The second development is a software platform that consists of a combination of watermarking and fingerprinting techniques with the objectives defined below:

- to help protect authorship and copyright of multimedia content,
- to provide content distributors and authors with a trusted system that allows the former to develop new business models while preserving the authorship rights of the latter,
- to offer the distribution of multimedia content via a Web platform, providing mechanisms for tracing dishonest users that illegally redistribute their content.

1.2.3 Objectives on mobile agents protection

The most difficult problem to solve in mobile agents environments is the attack from a platform against the agents. In this part, we analyse the use of software watermarking techniques as a possible solution to guarantee the integrity of mobile agents' execution. First we focus on protecting Intrusion Detection Systems (IDS) based on agents. The main strategies to achieve this goal are:

- use the Dynamic Graph Watermarking algorithm [23, 22] to store a watermarking into the structure of a graph which is created during the agent execution. When the agent's execution is modified, the graph is altered so the system can conclude that an attack has been performed.
- the use of Self-validating Branch-Based Software Watermarking algorithm [87] is studied in order to embed a matrix of marks in each transceiver of the IDS. Every time that an agent arrives to a host, this fingerprinting mark is monitored to verify that it is correct and conclude that the agent execution has not been modified.

Taking this an step further, we studied how to guarantee that the software is being executed correctly by a non-trusted host. In this sense, it is intended to:

- adapt the software watermarking scheme presented by Myles *et al.* in [87] to incorporate an external element called **sentinel** that controls the different execution stages of the agents,
- ensure the correct execution of the agent or, at least, detect illicit behaviours of the malicious system during the execution of the agent instead of detecting it once the agent has already been executed.

Finally, an infrastructure for detecting and punishing malicious hosts using mobile agent watermarking is presented. Specifically, the main objectives are:

- design a mobile agents infrastructure capable of detecting manipulation attacks performed by the host,
- propose a watermarking scheme of software that allows this detection,
- penalize hosts that are identified as malicious,
- present the results that demonstrate the usability of the solution.

1.3 Main contributions of this thesis

The results from the research carried out during the course of this thesis are mainly contained in 9 publications between *journals* included in JCR (2), *Lecture Notes in Computer Science* (5) as well as in various conferences.

1.3.1 Contributions related to fingerprinting codes and schemes

- IMPROVEMENT OF COLLUSION SECURE CONVOLUTIONAL FINGERPRINTING INFORMATION CODES
  International Conference on Information Theoretic Security (ICITS 2007), LNCS 4883, pp. 76 – 88, Springer-Verlag Berlin Heidelberg 2009
  DOI: 10.1007/978-3-642-10230-1_6
  Quality Index: CORE ranking C

- NEW CONSIDERATIONS ABOUT THE CORRECT DESIGN OF TURBO FINGERPRINTING CODES
  Esorics 2008, LNCS 5283, pp. 501 – 516, Springer-Verlag Berlin Heidelberg, 2008.
  DOI: 10.1007/978-3-540-88313-5_32
  Quality Index: CORE ranking A

- USE OF TURBO CODES WITH LOW-RATE CONVOLUTIONAL CONSTITUENT CODES IN FINGERPRINTING SCENARIOS
  IEEE Intl. Workshop on Information Forensics and Security - WIFS'11, 2011, pp. 1-6,
  DOI:10.1109/WIFS.2011.6123142
  Quality Index: h-index 16

1.3.2 Contributions related to secure e-commerce of multimedia content

- USING INFORMED CODING AND INFORMED EMBEDDING TO DESIGN ROBUST FINGERPRINTING EMBEDDING SYSTEMS
  11th International Conference on Knowledge-Based and Intelligent Information

& Engineering Systems (KES 2007), LNCS 4694, pp. 992 – 999, Springer-Verlag Berlin Heidelberg, 2007.
DOI:10.1007/978-3-540-74829-8_121
Quality Index: CORE ranking B

- TRAITOR TRACING OVER YOUTUBE VIDEO SERVICE—PROOF OF CONCEPT
Telecommunication Systems, Volume 45, Issue 1, pp. 47 – 60, Springer US, 2010-09.
DOI:10.1007/s11235-009-9236-z
Quality Index: Impact Factor (ISI) 0.705

1.3.3 Contributions related to mobile agents protection

- MAIS: MOBILE AGENT INTEGRITY SYSTEM. A SECURITY SYSTEM TO IDS BASED ON AUTONOMOUS AGENTS
SECRYPT 2007, Proceedings of the International Conference on Security and Cryptography, ICETE - The International Joint Conference on e-Business and Telecommunications, 2007.
Quality Index: CORE ranking B

- SECURING AGENTS AGAINST MALICIOUS HOST IN AN INTRUSION DETECTION SYSTEM
Critical Information Infrastructures Security, Second International Workshop, CRITIS 2007, LNCS 5141, pp. 94 – 105, Springer-Verlag Berlin Heidelberg, 2007.
DOI:10.1007/978-3-540-89173-4_9
Quality Index: CORE ranking C

- PROTECTION OF MOBILE AGENTS EXECUTION USING A MODIFIED SELF- VALIDATING BRANCH-BASED SOFTWARE WATERMARKING WITH EXTERNAL SENTINEL
Critical Information Infrastructures Security, Third International Workshop, CRITIS 2008, LNCS 5508, pp. 287 – 294, Springer-Verlag Berlin Heidelberg, 2009.
DOI:10.1007/978-3-642-03552-4_26
Quality Index: CORE ranking C

- AN INFRASTRUCTURE FOR DETECTING AND PUNISHING MALICIOUS HOSTS USING MOBILE AGENT WATERMARKING
Wireless Communications and Mobile Computing, Volume 11, Issue 11, pp. 1446 – 1462, John Wiley & Sons, Ltd., 2010.
DOI:10.1002/wcm.941
Quality Index: Impact Factor (ISI) 0.858

CHAPTER 2

STATE OF THE ART

In this chapter, the background of different aspects tackled during this thesis is introduced. The main important study areas in which the contributions of this document are bounded could be divided into:

- Watermarking techniques, that is, how a mark, which will contain some information, could be embedded into a digital document (mainly video and pieces of software) in order to guarantee a certain restrictions.

- Fingerprinting codes, that is, the alternatives for generating the marks embedded by means of watermarking techniques. The aim of these kinds of codes is to offer resilience against collusion attacks in which some users collude in order to generate an illegal copy of specific content.

- Iterative-decoding codes, which will be modified and adapted in order to be used as fingerprinting codes due to their reliability in noisy scenarios.

- Mobile agents, a study field in which the previous techniques have been applied in order to construct systems which are capable of prove the correct execution of an agent in host in real-time, that is, during his execution.

Even though the main ideas and definitions are presented in this chapter, more particular ones will be addressed during the next chapters in order to be more self-contained.

2.1 Digital Watermarking

The definition of watermarking is simple. A watermark is a message that is inserted into a document, and provides information about the author or the work itself. The reader can easily see that this vague definition applies to a great variety of fields. From the format of the original work; audio, video, text... to the topology of the incrusted mark, the spectrum of possibilities is endless.

2.1.1 Differences between watermarking and cryptography

Cryptography must not be confused with watermarking (the differences can be seen in the Figure 2.1). When a content is ciphered the user or client is forced to have a specific key to be able to view the original data. However, the intention of watermarking is not to avoid the user from displaying the content, but to prevent a dishonest user from removing the mark (a mark of property or a copyright) from the document. If we take an email as example, the encryption of it would prevent anybody except the authorized receiver from reading it. On the other hand, watermarking the email would prevent that a dishonest user deletes the signature and forwards the email changing the authorship of it.

The main technical objective of digital watermarking schemes is to build a robust and secure watermark. To accomplish this goal some basic requirements have to be accomplished. These requirements are:

- *Fidelity*: a watermark should be perceptually invisible, it should minimize degradation of the original content that's being marked.

- *Robustness*: a watermark should be difficult to remove. Specially, it should be resistant to the distortions caused by the typical signal processing mechanisms (conversion from analogue to digital, digital to analogue, re-quantification, re-compression...) and the generic distortions (rotation, translation, cropping...).

- *Capacity*: A watermarking system has to be capable of embedding a relative high amount of information. Being capable of storing a large quantity of information inside a mark makes the watermarking algorithm more flexible and versatile.

Some of this requirements are incompatible between them, for example it's not feasible to design a watermarking system that can provide a high robustness and capacity without introducing a high degree of distortion to the resulting document. On the other hand, an invisible and robust system is probably not able to offer a high information capacity. As a result of this constrains, the design of a watermarking schema

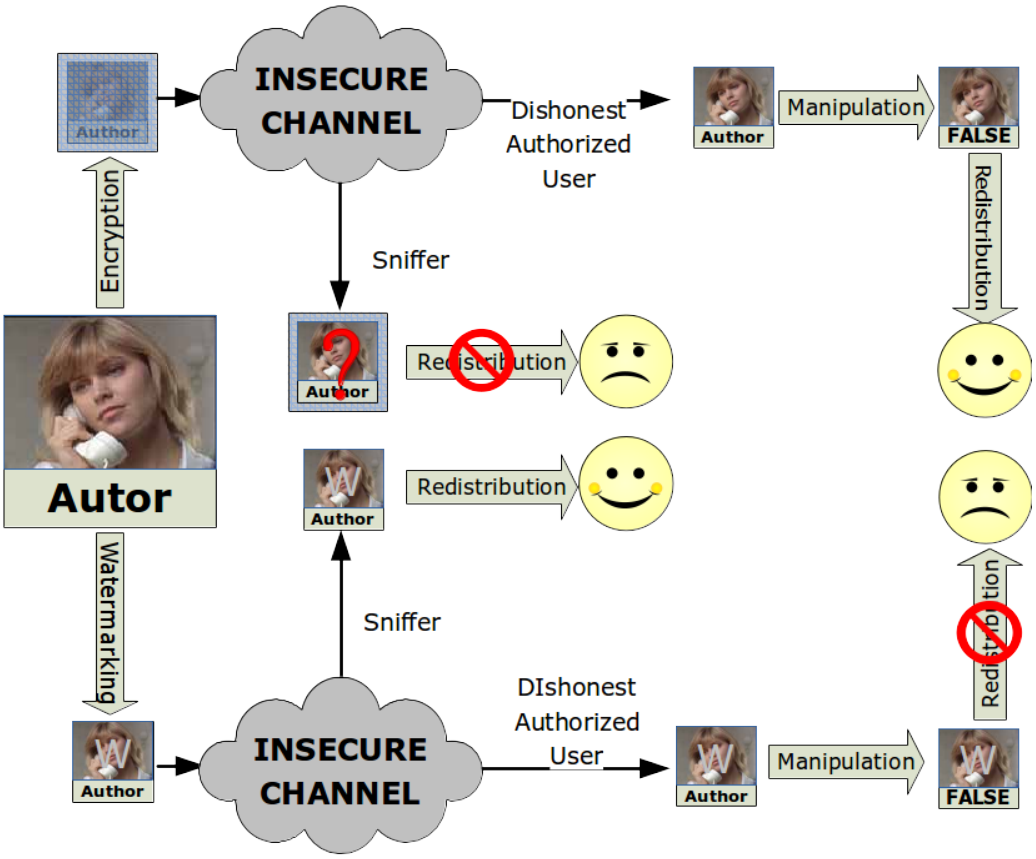


Figure 2.1: Differences between watermarking and cryptography

is a trade-off between all these presented parameters. Also, not all kinds of watermarking systems are designed to maximize this requirements, as an example the fragile watermarking is designed to detect any kind of modification made to a document, so it is very fragile, and when a document is modified the recovered mark is different from the original. This watermarking technique is based on a low robustness algorithm, in fact, the low robustness of the algorithm is what proves the validity of this method.

It exists other complementary requirements, and they vary accordingly to each application. The most relevant ones are the following:

- *Undetectable*: impossibility to prove the presence of a hidden message in a certain content. This concept is strongly tied to the statistical model of the original document. It is important to remark the capacity to detect the presence of a mark doesn't directly imply the possibility to remove it, but in some cases the possibility to apply a certain watermarking schema depends on this factor. Sometimes this requirement is mistaken with fidelity, it is important to remark that fidelity is based on perceptual measurements, on the other hand unde-

tactability is based on statistical methods.

- *Complexity*: the process of generating a mark should not be trivial, since a dishonest user may take advantage of this and could be able to generate a false mark. In that case, the system might not be able to differentiate pirate marks from original ones.

- *Access key*: embedded information should not be extracted, not even with attacks designed knowing the algorithm used in the embedding, the extraction of the mark and a marked document with its associated key. This property is exactly the same as the one applied in cryptographic systems.

- *Low error probability*: it is important to minimize the probability of detecting a mark incorrectly. It is also important to differentiate this situation from the one that happens when it is not possible to detect any mark. Recovering an erroneous mark may incriminate a honest user, since the recovered mark may match the mark of this user due to an error in the extraction. This presents a subtle issue, that has to the examined with caution.

- *Computational cost of the embedding and extraction*: usually the input files of this systems have a high size (a DVD movie), but it should be feasible to embed the mark in a short amount of time using appropriate and optimized algorithms. This factor is usually not take into account, since the computational power of devices is increased every day.

The application's scope of watermarking techniques is very large, and this thesis is centred on applying watermarking systems to video streams, in a similar way as still images, video streams are a set of coded images, usually using the DCT[1].

2.1.2 Classical image watermarking systems

Currently there is a large number of programs able to embed information inside of an image. Most of them work by adding the desired information in the less significant bits of every pixel [109]. When using this method, the information embedded in the system remains invisible to the human eye [70], but it is trivial for a third party to

[1]The Discrete Cosine Transform or abbreviated as DCT is a transformation based on the Discrete Fourier Transform but using only real numbers. Formally the Discrete Cosine Transformation is an invertible lineal function with an equivalent square matrix. The DCT used in the image domain is a 2D-DCT with a size of 8x8 pixels. The formal formulation is the following:

$$X_{k_1,k_2} \prec \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x_{n_1,n_2} cos\left[\frac{\pi}{N_1}\left(n_1+\frac{1}{2}\right)k_1\right] cos\left[\frac{\pi}{N_2}\left(n_2+\frac{1}{2}\right)k_2\right]$$

where X is the transformed matrix of x and $N_1 = N_2 = 8$ when using images.

detect and remove this information without introducing any distortion in the original image. Improved systems use a shared key between the sender and the receiver, and a pseudo-random cryptographic generator [101] to select the bits that will hold the encrypted data somehow [41].

It is logical to think that not all pixels in an image will be suitable for embedding the watermark: modifications made on pixels that belong to large areas of the same color (monochromes) or on sharp edges are more likely to cause visual distortion of the image. That is why some systems have algorithms designed to choose the best pixels for embedding extra information. This algorithms are based on the calculation of the variance of the luminance in the surrounding pixels (if the result of this difference is very high, it means the pixel belong to an edge, and if it is very low or null it means that the pixel belongs to a monochrome region). If a pixel passes this test, it is capable of holding the desired information in it's least significance bits.

This techniques can easily be neutralized by an attacker that the permissions to manipulate the marked image. As an example, almost every filtering process can modify the value of the less significant bits of an image, removing all the extra information and deleting the mark without affecting the quality of the image. A way to palliate this attack is to add an error correction code to the mark, or to add the same mark multiple times in the same image. The "Patchwork" algorithm, developed by Bender *et al.* [8], modifies the luminance of pairs of pixels selected in a pseudo-random way to embed a mark. A similar system was proposed by Pitas on [96]. Many similar techniques can be used to mark digital audio.

Another attack to this systems is to break the synchronization needed to detect the zones of the image where the information was embedded. Regarding images, it is possible to crop[2] a region of the image. On the audio field, an attack developed by Anderson *et al.* on [1] to accomplish a de-synchronization of the audio stream is described as removing random parts of the samples and duplicating the others. This attack introduces a jitter[3] of several microseconds that is capable of fooling the typical embedding systems.

Another way to attach information to an image is to embed this information inside the colour palette. A great number of the images distributed over the internet are coded using formats based on colour palette mechanism, like GIF or PNG. The advantage of this method is that it is easier to design a secure system for images with a certain degree of noise, like images coming from converted analogue sources (scanner, camera...). The main disadvantage is that the length of the mark is restricted to the size of the palette and not the size of the image.

On [71] it is suggested that a message might be hidden in a secure way on a colour

[2] *Cropping*: process where a portion of the image is removed from the original.

[3] *Jitter*: deviation in or displacement of some aspect of the pulses in a high-frequency digital signal

palette by shifting the order of the colours instead of actually changing the colours itself. This method doesn't change the visual aspect of the image perceived by the viewer, which is an advantage, but it's safety it's at least questionable, because most image processing programs order the palette in an specific order (usually regarding the luminance of the colours or their frequency). A randomly ordered palette would probably seem suspicious at least, and just opening the image and saving it again using any common image processing program would destroy the hidden message because the palette will be reordered.

More practical methods can obtain capacities that depend on the size of the image or the number of pixels. There is existing software that first decreases the depth of the colour palette of a GIF image to 128, 64 or 32 and then modifies the less significant bits of this palette to make it grow back to it's original size. Using this method it is possible to embed from one to three bits per pixel without creating much visual distortion. However, the newly created palette will have a similar group of colors, and the detection of this kind of marks is feasible because of this fact [65, 64].

One of the most popular methods to embed information in images based on a colour palette was proposed by Machado on [77]. In the proposed method, known as *EZ Stego*, the colour palette is first ordered according to its luminance. In the reordered palette, adjacent colours are very similar and *EZ Strego* embeds the message in a binary way using the less significant bits of the pixels that point to the colour palette.

The algorithm is based on the premise that adjacent colours in a luminosity ordered the palette are similar. However, as luminosity is a linear combination of the three colours R (red), G (green) and B (blue), eventually adjacent colours in a palette can represent different colours. To avoid this problem, Fridich proposes to embed this information in the parity bit (the calculation of the parity bit is as follows: $R + B + G \ mod \ 2$) found on adjacent colours [43]. Based on the colour of every pixel that is going to be used to embed information a search is performed to find similar colours on the palette until a colour with the desired parity is found. According to the author, it is realistic to assume that parity bits across a palette are distributed in a random way.

Currently, all these systems based on pixel or colour palette modifications have been put aside in favour of embedding the information in the transformed domain.

2.1.3 Common concepts in actual watermarking schemes

Usually the process of embedding a watermark follows four steps schematized in Figure 2.2. The first step is to transform the original document to a more appropriate domain to perform the embedding process. It is common to use the *Discrete Cosine Transform*, the *Fast Fourier Transform* or the *Discrete Wavelet Transform*. In this

project the *Discrete Cosine Transform* was used because most video compression systems are based on this transformation and the watermarking algorithm will be more efficient and easy to implement. The second step consist in coding the original message or the information to a mark that is suitable for embedding. The third step is the embedding of the mark inside the DCT coefficients, and the last step is the inverse DCT transformation, to recover the image.

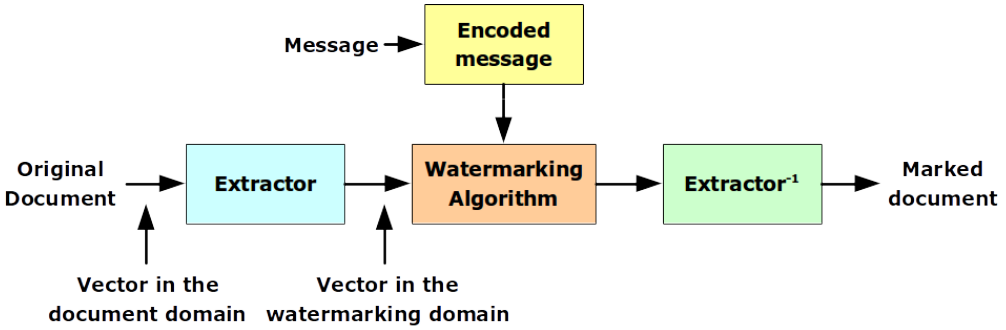


Figure 2.2: Generic embedding of marks

The process of recovering a watermark consists of three steps, as show in Figure 2.3. The first step is to convert the document to the domain where the mark was embedded and extract the mark from it. The second pass determines if the extracted information is a watermark, and if the result of this step is positive, it decodes the embedded message. Depending on the information available to perform the extraction, be can distinct between two different detection schemes:

- *Informed detection*: the original unmarked document is available.
- *Blind detection*: the decoder works without using the original document.

Usually, the availability of the original document during the detection process facilitates the decoding, and better results are obtained using informed detection.

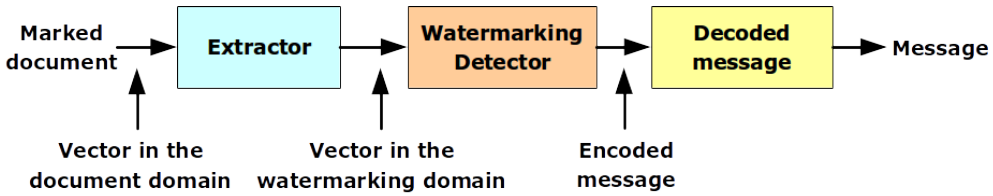
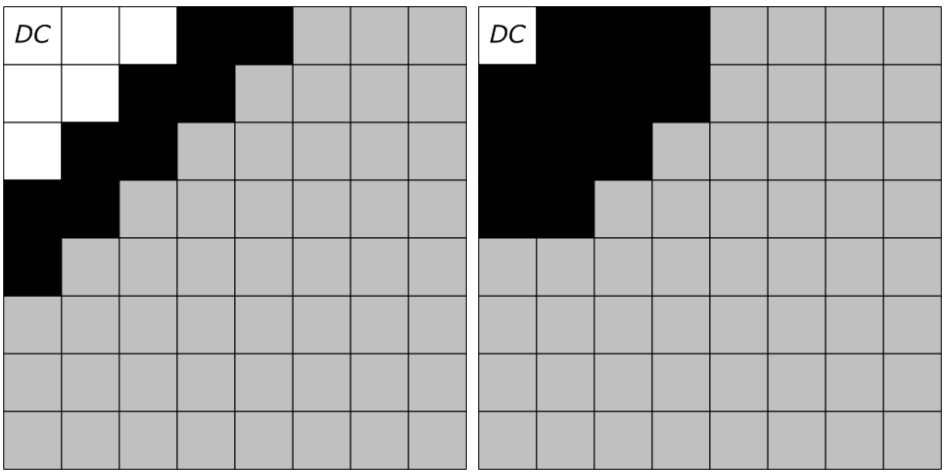


Figure 2.3: Generic mark extraction

As explained before, the marks are embedded directly into the DCT domain, modifying the coefficients of the transform. It is important to notice that not all the coefficients of the domain are modified, on the image field, the DCT matrix has sixty four

coefficients (structured in a 8×8 square matrix), but usually only some of them are used. The coefficients are not chosen randomly, usually the used coefficients are the ones where a small modification doesn't represent a big impact on image quality, but that they are enough important that they can not be removed from the image without a significant image quality loss. Different coefficients have been used for various studies, as an example the author of [27] uses the coefficients indicated in Figure 2.4a while the author of [82] uses the coefficients of Figure 2.4b.



(a) DCT coefficients DCT used in [27] (b) DCT coefficients DCT used in [82]

Figure 2.4: Black coefficients represents those used to embed the mark.

Regardless of the coefficients to be marked, the marking process applies the chosen transformation and selects the coefficients where the watermark information is going to be embedded. This positions can be permuted to increase the robustness of the algorithm.

2.1.4 Spread Spectrum Modulation

The Spread Spectrum modulation proposed by Cox *et al.* on [27] is one of the most famous watermarking schemes, probably because it was one of the first ones to embed the mark in the transformed domain (using the DCT). This article advocated that adding the mark to the most significant coefficients in the DCT would provide a most robust watermark. Also, this watermark should be generated using independent samples and equally distributed generated by a Gaussian distribution (N(0,1)). So any attempt to attenuate or remove the watermark would result in a severe degradation of the content.

This article defines the following notation: given a content to mark (*cover work*) $C = \{c_1, c_2, \cdots, c_N\}$, the length of the mark embedded $W = \{w_1, w_2, \cdots, w_N\}$ so that C can be converted to a marked vector $C_w = \{c'_1, c'_2, \cdots, c'_N\}$. On the proposal it is thought

that this marked document C_w is sent using a given channel that produces a set of attacks. Eventually, the document given to the detector is represented as C_w^*. One simple way to embed the mark could be

$$c_i' = c_i + \alpha \cdot w_i, \tag{2.1}$$

where α is the scaling parameter that controls the *trade-off* between the robustness and the visibility of the watermark. The equation 2.1 is always invertible, so given C_w^* and having C it is feasible to extract W^*. It seems logical that the next step is to put together W and W^* in order to determine if the extracted watermark is the same as the one that has been embedded. This evaluation is performed using the similarity function represented in the equation

$$sim(W, W^*) = \frac{W \cdot W^*}{\sqrt{W \cdot W^*}}. \tag{2.2}$$

The next step is to settle a threshold T so that if $sim(W, W^*) > T$, W is considered to be similar to W^* and the received document contains the mark W. On the other hand, if $sim(W, W^*) \leq T$, the document does not contain the mark W. The threshold level T has to be chosen having in mind that a value too high may cause false negatives (the detector incorrectly states that the document does not contain W), and a value too low may cause false positives (the detector incorrectly states that the document contains W when it actually does not).

Another way to calculate the similarity between W and W^*, which is usually more efficient regarding systems implementations is the following, defined in 2.3.

$$sim(W, W^*) = \frac{1}{N} W \cdot W^* = \frac{1}{N} \sum_i w_i \cdot w_i^* \tag{2.3}$$

If a bit of the message ($m = 0$ or $m = 1$) is going to be inserted inside a document c_0, a reference mark w_r of the same length as the document c_0. The watermark is defined using the following function

$$w_m = \begin{cases} w_r & \text{si } m = 1 \\ -w_r & \text{si } m = 0 \end{cases} \tag{2.4}$$

once we have obtained w_m it is inserted inside the original document c_0 as the equation 2.1 states.

Taking into account the noise injected during the process, the received content c_w can be defined as $c_w = c_0 + \alpha w_m + n$, where n is considered an additive Gaussian noise. The relation between the watermark and the noise can be measured using what the authors define as a Watermark-to-Noise relation (WNR),

$$WNR = 10 * \log_{10}\left(\frac{\delta_w^2}{\delta_n^2}\right). \tag{2.5}$$

To detect the watermark, a subtraction of the original document c_0 from c_w is performed to obtain w_m^*. Then the lineal correlation between w_m^* and w_r is calculated using the following function

$$z_{lc}\left(w_m^*, w_r\right) = \frac{1}{N}\left(\alpha w_m \cdot w_r + n \cdot w_r\right). \tag{2.6}$$

Taking into account that n has a Gaussian distribution, $n \cdot w_r$ can be considered negligible because the term $\alpha w_m \cdot w_r = \pm\alpha w_r \cdot w_r$ is going to be greater. So $z_{lc}\left(w_m^*, w_r\right) \approx \alpha w_r \cdot w_r$ if the watermark is $m = 1$ and $z_{lc}\left(w_m^*, w_r\right) \approx -\alpha w_r \cdot w_r$ if the watermark is $m = 0$. The detector should set the threshold T to $z_{lc}\left(w_m^*, w_r\right)$ and the result will be:

$$message = \begin{cases} 1 & \text{if } z_{lc}\left(w_m^*, w_r\right) > T \\ \text{no watermark} & \text{if } -T \leq z_{lc}\left(w_m^*, w_r\right) \leq T \\ 0 & \text{if } z_{lc}\left(w_m^*, w_r\right) < -T \end{cases}. \tag{2.7}$$

If the original document is divided in several segments and a vector is generated for each segment, various bits of the mark can be embedded. It is important to notice that the smaller the segment, the greater the distortion generated by the noise will be.

2.2 Digital Fingerprinting

During the last decade, the distribution and playback of digital images and other multimedia products has become a trivial issue to every computer user. Therefore, implementing satisfactory copyright protection systems has become almost a necessity. This topic is a challenging problem for the security research community. At first sight one can believe that data encryption already solves the problem. Unfortunately this is not the case, because encryption only offers protection as long as the data remains encrypted. The redistribution of digital products to unauthorized users becomes unavoidable once an authorized, but fraudulent, user decrypts it correctly.

A different approach to encryption is to discourage authorized users to misbehave. This idea lies at the core of fingerprinting techniques. The concept of fingerprinting was introduced by Wagner in [116] as a method to protect intellectual property in multimedia contents. The fingerprinting technique consists in making the copies of a digital object unique by embedding a different set of marks in each copy. Having unique copies of an object clearly rules out plain redistribution, but still a coalition of dishonest users can collude. A collusion attack consists in comparing the copies of the coalition members and by changing the marks where their copies differ, they can create a pirate copy in order to disguise their identities. Observe that in this situation it could be possible for the attackers to frame an innocent user. Thus, the fingerprinting

problem consists in finding, for each copy of the object, the right set of marks that help prevent collusion attacks.

In order to realize effective implementations of digital fingerprinting techniques, two important points must be contemplated: reliable embedding of the fingerprinting code and choosing the appropriate fingerprinting code. While the state of art in the first part has already been extensively presented in the section 2.1, this section will focus on the state of the art on the second part. It should be noted that this is a mathematical problem focused on the theory of codes and that will be treated from this perspective.

2.2.1 Properties of fingerprinting codes

Many efforts have been devoted to design new fingerprinting codes and to improve existing ones [12, 75, 102, 38, 6, 107]. In order to compare its performance, some measurement criteria, more or less objective, are needed, and to define measurement criteria, the properties to be measured must be defined. Therefore, the first thing to do is to define the basic properties that are desired for a good fingerprinting code:

- **High cardinality of *codebook*:** The code should be able to fit a large number of users. Although in some applications a small *codebook* can be appropriate (as it is the case of the Oscar's juries who only need a thousand copies for members of the Academy), in general, the size of *codebook* may be great. For example, the number of copies of a movie that are distributed to end users would be in the order of magnitude of millions.

- **Short codewords:** Embedding systems impose a limitation on the information capacity that they can embed in a particular document. If the codewords are short, the fingerprinting code will be adaptable to a greater number of applications than in the case they are long since there is not always a document with a very large capacity.

- **Easily traceable:** The more efficient the tracing algorithm is, the better the scheme of a particular fingerprinting code will be. Take into account that, in the case in which the detector has only limited computational capacity, the tracing algorithm is efficient enough to give decoding in a reasonable time.

- **Low false positives/false negatives probability:** It is obvious that an important property is that the probability that the tracing algorithm fails, that is return a false negative, has to be very low. Moreover, it is also desirable that, in the case of failing, the results will give obvious signs of having it produced an error instead of producing false positives, that is, to accuse innocent people.

As you can see with the naked eye, some properties conflict with each other quickly. For example, the length of the codewords and the size of *codebook*, since the shortest the codewords are, the lowest cardinality the code has. Another example is the relationship between the size of the codewords and the probability of error. Since in order to reduce the probability of error to a certain level the length of the codewords must be increased to add redundancy of some kind. In any case, these conditions usually lead to an increase in the computational cost of the tracing algorithms. Therefore, in many cases, when designing a fingerprinting code, it is necessary to consider the application to which it will be used.

2.2.2 Types of fingerprinting codes

In the context of digital fingerprinting, codes can be divided into two categories: *non-coded fingerprinting* and *coded fingerprinting*. Non-coded is a unique ID, serial number or any other information that is used to identify. The embedding process for each fingerprint involves the assignment of a single predefined mark for each fingerprint. An example is the use of orthogonal signals, in this case a different orthogonal signal is assigned to each fingerprint and in the detection the linear correlation is used [4] to decide which fingerprint is most likely among all possible. Although the implementation of the various proposals of this type of fingerprinting is usually simple and a good solution for small groups of users, they present the problem that their computational complexity in detection increases linearly with the number of users. Some remarkable articles where you can find more information about orthogonal fingerprints are [29, 118, 117].

In contrast, in the environments where coded fingerprinting is used, the original fingerprint (the serial number or the unique identifier) is encoded in order to obtain the real fingerprint. It is necessary to take into account the fact that, while the original fingerprints shouldn't have a structure or even of the same length, the real fingerprints are words made up of the same number of symbols of a particular alphabet. (From this point, when talking about fingerprint, this second type of fingerprinting will be referenced). One of the first publications on this type of fingerprinting appeared in 1995 in the hands of Dan Boneh and James Shaw [12]. In this publication the *Marking Assumption* was explicitly defined for the first time as well as a dimension of its length. Moreover, a method was proposed to build a safe binary code.

[4] The linear correlation between two vectors, v and w, it is the average of the product of its elements. $z_{lc}\sum_i v[i]\,w[i]$. It is used to verify the existence of the transmitted signal, w, in the signal received, v, by calculating $z_{lc}(w,W)$ and comparing the result with a certain threshold.

2.2.2.1 Marking Assumption

The *Marking Assumtion* is the essential principle for the design of a fingerprinting code, which is why it is crucial to discuss it in depth. In this section, the different versions presented in the literature will be introduced. In any case, it is good to present the different elements that take relevance in this environment. As a notation, we define that given a *word* w of length l-bits such that $w \in \Sigma^l$ and a set $I = \{i_1, \cdots, i_r\} \subseteq \{1, \cdots, l\}$ we indicate with $w\,|_I$ the word $w_{i_1} w_{i_2} \cdots w_{i_r}$ where w_i is the i-th letter of w. $w\,|_I$ will represent the *restriction* of w at the positions indicated by I.

First of all we will define a code (or *codebook*):

Definition 2.1 (Codebook [12])**.** *A set* $\Gamma = \{w^{(1)}, w^{(2)}, \cdots, w^{(n)}\} \subseteq \Sigma^l$*, where* Σ *will denote some alphabet of size s, will be called an* (l, n, s)*-code. The codeword* $w^{(i)}$ *will be assigned to user* u_i*, for* $1 \leq i \leq n$*. We refer to the set of words in* Γ *as the* ***codebook****.*

Once the concept of *codebook* is defined, the next important element will be the *undetectable positions*. Conceptually we define a coalition as a group of users whose members try to attenuate or eliminate the fingerprints of their copies in order to generate a new copy with a fingerprint that is different from its own and that it can not incriminate them. Actions can go from to make the average between the copies and distribute the result to much more sophisticated techniques. We can consider as *undetectable positions* those positions marked in a document where all members of the attackers' coalition have the same value and, therefore, they can not discriminate against the positions not marked in the document. A more formal description is the one given by Boneh-Shaw in [12]:

Definition 2.2 (Undetectable Position [12])**.** *Let* $\Gamma = \{w^{(1)}, w^{(2)}, \cdots, w^{(n)}\}$ *be an (l,n,s)-code and* $C = \{u_1, u_2, \cdots, u_c\}$ *be a coalition of c-traitors. Let position* $i \in \{1, \cdots, l\}$ *be* ***undetectable*** *for C, if the words assigned to users in C match in i'th position, that is* $w_i^{(u_1)} = \cdots = w_i^{(u_c)}$*.*

On the different definitions of *Marking assumptions* a consensus exists on this:

Definition 2.3 (Marking Assumption)**.** *Let* $\Gamma = \{w^{(1)}, w^{(2)}, \cdots, w^{(n)}\}$ *be an (l,n,s)-code,* $C = \{u_1, u_2, \cdots, u_c\}$ *a coalition of c-traitors and* $\Gamma(C)$ *the* ***feasible set*** *of C. The coalition C is only capable of creating an object whose fingerprinting lies in* $\Gamma(C)$*.*

The problem lies in how this **feasible set** is defined. In the literature there are up to 6 different definitions for this term.

Definition 2.4 (Feasible Set according to [18, 104])**.** *Let* $\Gamma = \{w^{(1)}, w^{(2)}, \cdots, w^{(n)}\}$ *be an* (l, n, s)*-code and* $C = \{u_1, u_2, \cdots, u_c\}$ *be a coalition of c-traitors. We define the* ***feasible***

set $FS(C;\Gamma)$ *of* C *and* Γ *as*

$$FS(C;\Gamma) = \{x = (x_1, \cdots, x_l) \in \Sigma^l \mid x_j \in \{w_j \mid w \in C\}, 1 \leq j \leq l\}.$$

In [120] the case in which the coalition knows the entire alphabet is considered. And therefore can use any symbol of the alphabet in the positions detected, so a new definition could be the following:

Definition 2.5 (Feasible Set according to [120])**.** *Let* $\Gamma = \{w^{(1)}, w^{(2)}, \cdots, w^{(n)}\}$ *be an* (l, n, s)*-code and* $C = \{u_1, u_2, \cdots, u_c\}$ *be a coalition of* c*-traitors. We define the* ***feasible set*** $FS(C;\Gamma)$ *of* C *and* Γ *as*

$$FS(C;\Gamma) = \{x = (x_1, \cdots, x_l) \in \Sigma^l \mid x_j \in w_j, 1 \leq j \leq l\}$$

where

$$w_j = \begin{cases} \{w_j^{(u_1)}\} & w_j^{(u_1)} = \cdots = w_j^{(u_c)} \\ \{\Sigma\} & otherwise \end{cases}$$

Moreover, if we include in the previous definitions the possibility that the attackers could manage to erase the detected positions, we could reformulate them as:

Definition 2.6 (Feasible Set according to [120])**.** *Let* $\Gamma = \{w^{(1)}, w^{(2)}, \cdots, w^{(n)}\}$ *be an* (l, n, s)*-code and* $C = \{u_1, u_2, \cdots, u_c\}$ *be a coalition of* c*-traitors. We define the* ***feasible set*** $FS(C;\Gamma)$ *of* C *and* Γ *as*

$$FS(C;\Gamma) = \{x = (x_1, \cdots, x_l) \in (\Sigma \cup \{?\})^l \mid x_j \in w_j, 1 \leq j \leq l\}$$

where

$$w_j = \begin{cases} \{w_j^{(u_1)}\} & w_j^{(u_1)} = \cdots = w_j^{(u_c)} \\ \{w_j^{(u_i)} \mid 1 \leq i \leq c\} \cup \{?\} & otherwise \end{cases}$$

and ? *denotes an erased position.*

And in the case in which the coalition knows the entire alphabet:

Definition 2.7 (Feasible Set according to [12])**.** *Let* $\Gamma = \{w^{(1)}, w^{(2)}, \cdots, w^{(n)}\}$ *be an* (l, n, s)*-code and* $C = \{u_1, u_2, \cdots, u_c\}$ *be a coalition of* c*-traitors. We define the* ***feasible set*** $FS(C;\Gamma)$ *of* C *and* Γ *as*

$$FS(C;\Gamma) = \{x = (x_1, \cdots, x_l) \in (\Sigma \cup \{?\})^l \mid x_j \in w_j, 1 \leq j \leq l\}$$

where

$$w_j = \begin{cases} \{w_j^{(u_1)}\} & w_j^{(u_1)} = \cdots = w_j^{(u_c)} \\ \{\Sigma\} \cup \{?\} & otherwise \end{cases}$$

and ? *denotes an erased position.*

Safavi-Naini and Wang proposed in [98] a further step on the proposal of the definition 2.6 allowing to add erased symbols to the undetectable positions.

Definition 2.8 (Feasible set according to [98])**.** *Let* $\Gamma = \{w^{(1)}, w^{(2)}, \cdots, w^{(n)}\}$ *be an* (l, n, s)*-code and* $C = \{u_1, u_2, \cdots, u_c\}$ *be a coalition of* c*-traitors. We define the* ***feasible set*** $FS(C;\Gamma)$ *of* C *and* Γ *as*

$$FS(C;\Gamma) = \{x = (x_1, \cdots, x_l) \in (\Sigma \cup \{?\})^l \mid x_j \in w_j, 1 \leq j \leq l\}$$

where $w_j = \left\{\{w_j^{(u_i)} \mid 1 \leq i \leq c\} \cup \{?\}\right\}$ *and* ? *denotes an erased position.*

Finally, the last proposal comes from Guth and Pfitzmann in [49]. In this proposal a set of random errors are added to the code word and C can generate any word.

Definition 2.9 (Feasible set according to [49])**.** *Let* $\Gamma = \{w^{(1)}, w^{(2)}, \cdots, w^{(n)}\}$ *be an* (l, n, s)*-code and* $C = \{u_1, u_2, \cdots, u_c\}$ *be a coalition of* c*-traitors. We define the* ***feasible set*** $FS(C;\Gamma)$ *of* C *and* Γ *as*

$$FS(C;\Gamma) = \{x = (x_1, \cdots, x_l) \in (\Sigma \cup \{?\})^l \mid x_j \in w_j, 1 \leq j \leq l\}$$

where $w_j = \{\{\Sigma\} \cup \{?\}\}$ *and* ? *denotes an erased position, that is* w_j *could be any symbol in the alphabet or an erased position.*

To illustrate the differences, the following assumption is made: Σ such $\Sigma = \{A, B, C, D, E, F\}$ is available and words of length $l = 4$ are generated. A coalition of 3 users is defined that their codewords are:

$$\begin{array}{lcccc} x^{(u_1)}: & A & C & F & B \\ x^{(u_2)}: & A & C & E & A \\ x^{(u_3)}: & A & B & D & F \end{array}$$

The content of the *Feasible Set* according to the various definitions would be shown in the table 2.1.

In any case, this thesis focuses on the case of the definition 2.6, which in the binary case is equivalent to that made by Boneh-Shaw in its article and that corresponds to the definition 2.7. It is considered that an attacking coalition is only able to detect those positions in which the marks for the various members of the coalition differ. So, the attackers can choose between selecting the value of any of them or damaging the position in order to cause an erasure. On the other hand, in a real case, this assumption is considered the most reasonable since, although the alphabet can be known, the attackers do not have to know how this one is codified to the watermarking level. Finally, in video environments, positioning in the suppose of [49] would be unrealistic

***Feasible Sets* according to the different definitions**				
Definition	w_0	w_1	w_2	w_3
$FS(C;\Gamma)$ def. 2.4:	A	$\{B,C\}$	$\{D,E,F\}$	$\{A,B,F\}$
$FS(C;\Gamma)$ def. 2.5:	A	$\{\Sigma\}$	$\{\Sigma\}$	$\{\Sigma\}$
$FS(C;\Gamma)$ def. 2.6:	A	$\{B,C\}\cup\{?\}$	$\{D,E,F\}\cup\{?\}$	$\{A,B,F\}\cup\{?\}$
$FS(C;\Gamma)$ def. 2.7:	A	$\{\Sigma\}\cup\{?\}$	$\{\Sigma\}\cup\{?\}$	$\{\Sigma\}\cup\{?\}$
$FS(C;\Gamma)$ def. 2.8:	$\{A\}\cup\{?\}$	$\{B,C\}\cup\{?\}$	$\{D,E,F\}\cup\{?\}$	$\{A,B,F\}\cup\{?\}$
$FS(C;\Gamma)$ def. 2.9:	$\{\Sigma\}\cup\{?\}$	$\{\Sigma\}\cup\{?\}$	$\{\Sigma\}\cup\{?\}$	$\{\Sigma\}\cup\{?\}$

Table 2.1: Different set ups of the Feasible Set depending on the different definitions present in the literature. $\{\Sigma\}$ denotes any symbol in the alphabet Σ and ? denotes an erased position

since inserting a noise in all the markable positions of a video with sufficient intensity to alter the mark would cause a distortion of the sequence sufficiently important to make the video unusable.

2.2.2.2 Classification of traceability codes according their resistance to collusion attacks

This section will give the definitions of the types of collusion secure codes. In order to make an accurate description, the concept of descendants must be defined before, that is to say, given a Feasible Set which words of the code can be formed with it. A formal definition would be:

Definition 2.10 (Descendants of C)**.** *Let $\Gamma = \{w^{(1)},\cdots,w^{(n)}\}$ is a (l,n,s) - code and $FS(C;\Gamma)$ is the Feasible Set of a coalition C of at most c user of Γ then, we define the set of descendants of C, denoted $desc(C)$, as*

$$desc(C) = \left\{ y \in \{\Sigma \cup \{?\}\}^l \mid y_i \in \{x_i \mid x \in FS(C;\Gamma)\}, 1 \le i \le l \right\}$$

And, from this definition, it is defined the k-descendant code as the code formed by all descendants of all coalitions of up to k users. Formally,

Definition 2.11 (k-descendant code)**.** *Let k be a positive integer. For an (l,n,s)-code Γ, define the k-descendant code, denoted $desc_k(\Gamma)$, as follows:*

$$desc_k(\Gamma) = \bigcup_{C\subseteq\Gamma, |C|\le k} desc(C)$$

The first type of codes are c-frameproof, that is, given a collusion of c users, they can not generate a codeword for a user that does not form part of the coalition. These codes were defined in [12]. Formally,

Definition 2.12 (c-frameproof)**.** *Let Γ be an (l,n,s)-code, it is c-frameproof or c-FP if for any $x \in desc(C)$ such that $C \subset \Gamma$, $c \ge |C|$, $x \in FS(C;\Gamma)\cap\Gamma \Longrightarrow x \in C$*

A refinement of these types of codes are called c-secure frameproof. These codes were presented in [105] and guarantee that two different coalitions of c different users, without any user in common, can not generate the same codeword. The formal definition would be:

Definition 2.13 (c-secure frameproof)**.** *Let Γ be a (l,n,s)-code, it is c-secure frameproof or c-SFP if for any $C_1, C_2 \subseteq \Gamma$ such that $|C_1| \leq c$, $|C_2| \leq c$ and $C_1 \cap C_2 = \emptyset$, we have $FS(C_1;\Gamma) \cap FS(C_2;\Gamma) = \emptyset$.*

So far, the codes presented in definitions 2.12 and 2.13 assure that a C coalition of no more than c users can not frame an innocent user even though they can generate a word that is not part of the code.

When a word that is not part of the code is detected in an illegal distribution, the distributor can realize that this is a pirated copy but this code does not allow finding a member of the coalition C that is, the ultimate goal that the distributor pursues. The following three codes defined in 2.14, 2.16 and 2.17 allow tracing to determine at least one member of the coalition that has generated the pirate word, whether it is a pirate in the sense of being the result of a collusion attack or as a user who has redistributed his copy.

The first type of codes have the Identifiable Parent Property (IPP), meaning that for each word x that belongs to the descending code, a code word w belonging to Γ can we found that is common in all coalitions that can form x. This type of code was presented in [59]. Formally,

Definition 2.14 (Identifiable parent property)**.** *Let Γ be an (l,n,s)-code and $C_i \subseteq \Gamma$ where $|C_i| \leq c$, we say that it has the identifiable parent property for coalitions of at most c parents (c-identifiable parent property) or c-IPP if for any $x \in desc_c(\Gamma)$ we have*

$$\bigcap_{i|x \in desc(C_i)} C_i \neq \emptyset$$

An evolution about this property would be the *traceability* discussed in depth in [18], where it was defined, and [106]. The basic property of these codes is that, with respect to a pirated word, the codewords of the users that do not belong to the coalition have a Hamming distance[5] greater than at least one of the colluding codewords. The idea is that not only a coalition can not frame an innocent but also, given a pirated word, the code word that is closest in terms of Hamming's distance will be part of the coalition. In any case, we first define the index concept of undetectable bits such as:

[5]The distance of Hamming between two words of equal length is the number of positions in which the symbols are different.

Definition 2.15 (Index to undetectable bit)**.** *Let $x = \{x_1, \cdots, x_l\}$ and $y = \{y_1, \cdots, y_l\}$ such that $x, y \in \Sigma^l$, we define the Index to undetectable bit between x and y as $I(x,y) = \{i | x_i = y_i\}$.*

Once defined, this concept will be used in the definition of the traceability codes:

Definition 2.16 (Traceability code)**.** *Let Γ be a (l,n,s)-code and $C \subseteq \Gamma$ where $|C| \leq c$, we say that Γ has traceability property for coalitions of at most c parents or c-TA if for any $x \in desc(C)$ exists at least one code word $y \in C$ such that $|I(x,y)| > |I(x,z)|$ for any $z \in \Gamma \setminus C$ where $\Gamma \setminus C$ represents the set Γ minus the subset C.*

Going one step further, we reach the ideal codes defined in [12] and receive the name of *totally c-secure*. Formally, considering the feasible set defined in 2.7,

Definition 2.17 (totally c-secure)**.** *Let Γ be a (l,n,s)-code, this code is totally c-secure if there exists a tracing algorithm A satisfying the following condition: if a coalition C of at most c users generates a word x then $A(x) \in C$.*

The tracing algorithm A on input x must output a member of the coalition that generated the word x. Unfortunately, when $c \geq 2$, totally c-secure codes do not exist. This was formulated as the theorem 2.1 and proof by Boneh and Shaw in [12].

Theorem 2.1. *For $c \geq 2$ and $n \geq 3$ there are no totally c-secure (l,n)-codes.*

However, Boneh i Shaw, also in [12], propose codes with a small relaxation of conditions with respect to *totally c-secure*, c-secure with a probability of error ϵ. The formal definition would be the following:

Definition 2.18 (c-secure with ϵ-error)**.** *Let Γ be a (l,n,s)-code, it is c-secure with ϵ-error probability if there exists a tracing algorithm A satisfying the following condition: if a coalition C of at most c users generates a word x then*

$$Pr[A(x) \in C] > 1 - \epsilon$$

In [104] the relationship between these different codes is studied and the proposition 2.1 is shown in [120]. Another study on the relationship between the IPP codes and TAs can be found in [55].

Proposition 2.1. *Let Γ be a (l,n,s)-code, then, we have the following:*

1. *Γ is a c-IPP if Γ is a c-TA.*
2. *Γ is a c-SFP if Γ is a c-IPP.*
3. *Γ is a c-FP if Γ is a c-SFP.*

Keep in mind that the reverse implications do not have to be met and, in fact, in [104] there are several counter-examples.

2.2.3 Comparison of the most significant existing fingerprinting schemes

The table 2.2 shows some of the most relevant existing fingerprinting algorithms. The first five schemes are adaptable for coalitions of any size, while the other two are explicitly designed for coalitions of 2 or 3 users respectively.

Authors	Users	Attackers	Error	Length of the code
Boneh & Shaw [11]	N	c	ϵ	Inner code (n,d), Outer code (L,N), $L = 2clog_e(2N/\epsilon)$, $d = 2n^2 log_e(4nL/\epsilon)$, $l = Ld(n-1)$
Lindkvist [75]	N	c	0	$l = \frac{\binom{N}{c}}{\binom{q-1}{c}}$, $c < q < N$
Staddon, Stinson & Wei [104]	N	c	ϵ	$N = q^{\lceil \frac{l}{c^2} \rceil}$, $l \leq q+1$
Fernández & Soriano [38]	N	2	ϵ	$(2^r-1, r, 2r-1)$ DH(r) $(n, \lceil \frac{n}{4} \rceil), n - \lceil \frac{n}{4} \rceil) + 1$ RS code $l = (2^r-1)n$
Sebe & Domingo-Ferrer [102]	N	3	ϵ	Scatter code (d,t), n-Dual Hamming Code $l = (2^n-1)(2t+1)d$
Barg, Blakley & Kabatiansky [6]	N	c	ϵ	$l = O(2^c c \log c \log N)$
Cotrina & Fernandez [25]	N	c	ϵ	$l = O(c^6 \log(c/e) \log N)$
Tardos [107]	N	c	ϵ	$l = 100c^2 \ln \frac{N}{\epsilon}$

Table 2.2: Characteristics of some fingerprinting codes.

Collusion secure fingerprinting principles derive from the seminal work of Boneh and Shaw in [11], where they present a collusion n-secure code with ϵ error, where n represents the total number of authorized users, and ϵ denotes the probability of not identifying a traitor user. Since in practical scenarios we will rarely have to face a traitor coalition that consists of *all* authorized users, c-secure codes, with $c < n$ will be sufficient for our purposes. The code of Boneh and Shaw is a c-secure binary code with probability of error ϵ able to serve a large number of users with the ability to withstand high-probability coalitions of many users. Even so, the price paid for it is a very large code length. This code is a concatenated code where the codewords are generated by the use of codewords of an internal code in order to code the symbols of the external code. The *inner code* is a binary code with length l and size n symbolized as $\Gamma_0(n,d)$ and compounded by n codewords of length $l = d(n-1)$, $\Gamma_0(n,d)$ is a repetition code, the probability of error ϵ can be adjusted by choosing a different d (known as a repetition ratio). The outer layer can be a random (N,L)-code with N

codewords on a size alphabet of n (the number of codewords for the internal code) and each code word with length L. The resulting construction is a $(N, L(n-1)d)$-code. As can be seen in the table 2.2, the length of the codewords is enormous and, in most cases, cannot be implemented for real applications. Moreover, due to the random nature of the outer code in the concatenation, this construction does not have any polynomial-time (in the code length) decoding algorithm, thus limiting their practical applications for large values of N.

The Staddon, Stinson and Wei [104] code is a c-secure q-ary fingerprinting code. It is built using a Reed-Solomon code and is a code with traceability properties. The length of the code is significantly lower than the Boneh and Shaw code.

The Lindkist code (also known as a Gossip code), proposed in [75], is a q-ary fingerprinting code, and has the IPP property. Ravi S. Veerubhotla et al. in [110] give a general construction in order to achieve the minimum code length possible according to the specification for Gossip codes, in terms of alphabet size, number of codewords and coalition size. The advantage of the Gossip codes is that they can accuse a traitor deterministically while the previous constructions were probabilistic. In any case, the size of the codewords is still a problem for real applications, as can be seen in the table 2.2.

The code presented by Sebe and Domingo-Ferre on [102] is specifically designed against coalitions of three users ($c = 3$), the construction contains two layers, the internal code known as *scattering code* and the external code that is a dual Hamming code. The *scattering code* (d, t) is a binary code that consists of $2t$ codewords of length $(2t+1)d$. The final code can represent up to N users with codewords of length $(N-1)(2t+1)d$. For a certain level of security (or probability of error ϵ), the codes can be much shorter than those obtained by the proposal of Boneh and Shaw while the number of users is moderated (less than 2^{16}).

The code of Fernandez and Soriano [38] is specifically designed for coalitions of size $c = 2$, it is a binary code with two layers, the internal code is $(2,2)$-separable (they use a dual binary Hamming code), while the external code is a Reed-Solomon code with the IPP property. The advantage offered by this code is that the construction allows the use of an efficient decoding algorithm that corrects above the code's corrective capacity (a simplified version of the Chase algorithm [14] for the internal code and Koetter-Vardi's soft-decision list decoding algorithm presented in [69].

In another breakthrough work Barg et al. in [6], constructed c-secure fingerprint-

ing codes, with error probability less than $exp(-\Omega(\log n))$, of length $0(2^c \log n)$ and decoding algorithm of complexity $poly(\log n)$. Their schemes are based on the concatenation of Algebraic Geometric Code (AG) codes, and binary (c,c)-separable codes. This construction improves in many ways the construction of Boneh and Shaw. It possesses an efficient decoding algorithm. Moreover, for a fixed c, it is asymptotically better in n and ϵ.

Cotrina and Fernandez presented [25] a new construction that is a mixture of the codes in [11] and [6]. They combine asymptotically good AG codes with Boneh-Shaw codes. The error probability of the concatenated construction is $O(1/n) = exp(-\Omega(\log n))$, with length of order $O(c^6 \log c \log n)$, and decoding algorithm of complexity $poly(\log n)$.

Tardos, building upon Boneh and Shaw's codes also used randomization in determining the codewords [107]. By representing the code as a binary matrix in which each row corresponds to a codeword, there is a binary distribution for each column, and each entry is a given column is selected to be 0 or 1 according to the corresponding distribution. By cleverly choosing those distributions and the tracing algorithm, the length of Tardos codes is $O(c^2 \log(n/\epsilon))$, roughly the square root of that in the Boneh-Shaw codes. This code length is optimal, within a constant factor, for sufficiently small values of ϵ.

2.3 Codes with iterative decoding

As Ken Gracie and Marie-Hélène Hamon state in [48], getting a generic definition for the turbo and "turbo-like" codes is quite complicated. Even so they share three very identifying features:

- **Composite Structure:** The information bits are encoded with multiple low-complexity *constituent* or *component* codes. Each constituent code may or may not involve all of the information bits.

- **Interleaving:** The information bits are reordered or permuted between encoders.

- **Soft iterative decoding:** The component codes are decoded multiple times and soft reliability estimation derived from each code are used to improve the decoding of the other component codes.

By means of using the idea of concatenating simple codes to obtain better codes, iterative-decoding codes use permuted versions of the same information encoded

with reasonable simple codes. Even the simplicity of the constituent codes, the resulting codes have characteristics really close to the Shannon limit.

The iterative decoding begins by decoding the constituent codes individually, simultaneously or not, using the input retrieved from the channel and a certain *a priori* information (in the first step it is assumed that the bits can take either values 0 or 1 with a the same probability so this information will cost $\frac{1}{2}$). The information on the symbols obtained from this first decoding is shared with the other decoders and so repeatedly until it is no longer improved on the result. In other words, the information obtained from the channel and the one provided by the other decoders is used to improve the decoding of a given constituent code, and thus the joint decoding is improved. The decoder not only provides information about the value of each bit but also informs about the veracity of this value.

Another responsible of the increase in performance presented by these type of codes is the use of information called "soft" instead of "hard" decisions, that is, the use of real numbers, the values which represent the reliability of the estimated symbol, instead of using only the symbols. Iterative decoding requires that the algorithms that decode component codes use soft information both in input and output (known as soft-in soft-out algorithms or SISO). This process of exchanging soft information between the various parts that make up the decoder is known as a message crossing (*message passing*) or a propagation of trust (*belief propagation*) and was introduced in [44] by Gallager. The spread of trust is based on passing messages locally within the structure of the code, that is, estimations of a given symbol are updated based on the symbols that are close to the structure of the code. The information of a given symbol is propagated throughout the structure of the code. In the probabilistic environment, the *likelihood ratios* or LR of each bit d are defined in order to propagate this information. The formal definition for the binary data symbols could be expressed as:

$$\ell^k = \frac{Prob\{d = 0 \mid \mathbf{y}, C\}}{Prob\{d = 1 \mid \mathbf{y}, C\}} = \ell_n^k \cdot \prod_i \ell_{ei}^k, \tag{2.8}$$

where $\mathbf{y}$ is the decoder input, C refers to the code structure, ℓ_n^k denotes the contribucion due to the decoder inputs (the *intrinsic* information), ℓ_{ei}^k denotes the contribution due to the constituent codes (the *extrinsic* information) and i denotes the number of constituent decoding operations. Often, the constituent decoders operate in the log domain. In this case, the shared information between the decoders is the Log-Likelihood Ratio (LLR), expressed as

$$L^k = ln\frac{Prob\{d = 0 \mid \mathbf{y}, C\}}{Prob\{d = 1 \mid \mathbf{y}, C\}} = L_n^k + \sum_i L_{ei}^k. \tag{2.9}$$

2.3.1 Turbo Codes

Turbo codes were introduced in 1993 by Berrou, Glavieux and Thitimajashima [9], [10]. In their research, they reported extremely impressive results for a code with a long frame length. The main idea is an extrapolation from Shannon's theory of communication. Shannon [103] shows that an ultimate code would be one where a message is sent infinite times, each time shuffled randomly, but this requires infinite bandwidth so this schema is unpractical. The contribution of turbo codes is that sending the information infinite number of times is not really needed, just two or three times provides pretty good results.

2.3.1.1 Turbo Coding

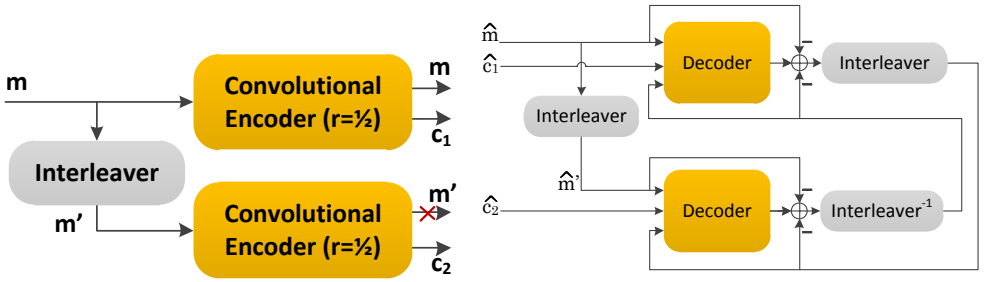
The most common turbo encoder consists of parallel concatenation of some Recursive Systematic Convolutional encoders (RSC) [9], each with a different interleaver, working on the same information. The purpose of the interleaver is to offer to each encoder an uncorrelated version of the information. The usual configuration consists of two identical convolutional encoders with rate $1/2$ and a pseudo-random interleaver, π. As shown in Figure 2.5a, this is called a Parallel Concatenated Convolutional Code (PCCC).

The input bits m are grouped in sequences whose length N is equal to the size of the interleaver. The sequence m' is obtained as the result of the interleaving process. The first encoder receives the sequence m and produces the pairs (m, c_1) and the second encoder receives the sequence m' and produces the pairs (m', c_2). Since both encoders are systematic encoders $m' = \pi(m)$ and, as π is known by the decoder, only (m, c_1, c_2) will be transmitted. The rate of this encoder is $1/3$ but it can be increased by puncturing by $1/2$.

2.3.1.2 Turbo Decoding

Turbo decoding is based on an iterative process to improve performance and it uses, as a basic decoder unit, a Soft-Input Soft-Output algorithm. The block scheme of a common turbo decoder is shown in figure 2.5b.

First of all, the sequence encoded by the first encoder is decoded by the first decoder. As a result, this decoder returns soft information, that is to say, an estimation about which were the values of the bit in the original sequence and how likely is this estimation for each bit. This information is usually called extrinsic information. The



(a) Common turbo encoder (b) Common turbo decoder

Figure 2.5: Dual Turbo Encoder/Decoder with ratio $r = \frac{1}{3}$.

extrinsic information of the first decoder is interleaved in the same manner that the input bits had been interleaved in the turbo encoder before they are applied to the second encoder. The next step is to send this interleaved information to the second decoder. This decoder takes the extrinsic information of the first decoder into account when it decodes the sequence encoded by the second encoder and gives a new estimation about the original values. This process is repeated several times depending on the performance that is required of the system. On the average, 7 or 8 iterations give adequate results and no more 20 are ever required.

There are some algorithms that can be modified to use as a turbo decoder component but the ones most used are the Soft Output Viterbi Algorithm [50, 52] and the BCJR [2] or Maximum A-posteriori Probability (MAP) algorithm. SOVA is a combination of iterative decoding with a modified form of Viterbi decoding and it maximizes the probability of a sequence. On the other hand, MAP maximizes the output probability based on some knowledge of the input *a priori* probabilities and soft output from the demodulator.

2.3.1.3 The Max-Log-MAP Algorithm

The MAP algorithm gives, for each decoded bit (b_i), the probability that this bit was +1 o -1, given the received symbol sequence $\mathbf{Y}$. This is equivalent to finding the *a posteriori* log-likelihood ratio (LLR):

$$L(b_i \mid \mathbf{Y}) = \ln\left(\frac{P(b_i = +1 \mid \mathbf{Y})}{P(b_i = -1 \mid \mathbf{Y})}\right) \tag{2.10}$$

The Max-Log-MAP algorithm is a simplified version of the MAP algorithm by transferring these equations into the logarithmic domain and using the approximation:

$$\ln\left(\sum_i e^{a_i}\right) \approx \max_i(a_i) \tag{2.11}$$

Forward, backward and transition state metrics, α, β and γ, are calculated as shown below:

$$\begin{aligned} \alpha_i(u) &\simeq \max_{u'}\{\alpha_{i-1}(u') + \gamma_i(u', u)\} \\ \beta_{i-1}(u') &\simeq \max_{u}\{\beta_i(u) + \gamma_i(u', u)\} \\ \gamma_i(u', u) &\simeq C\ln(e^{(b_i L(b_i))/2} e^{\frac{L_c}{2}\sum_{k=1}^{n}(y_{ik}x_{ik})}) \end{aligned} \tag{2.12}$$

where C is a constant value, b_i is the information bit, $L(b_i)$ is the *a priori* information for each constituent decoder, the constant $L_c = 2\frac{E_b}{\sigma^2}$ is a measure of the Signal-to-Noise Ratio (SNR) in the channel and, finally, y_{ik} and x_{ik} are the received and transmitted bits.

To determine the reliabilities LLR in Max-Log-MAP algorithm, in contrast to the MAP algorithm, only the most likely paths at time i are considered. This approximation is one of the reasons for the sub-optimal performance of the Max-Log-MAP algorithm compared to the MAP algorithm.

The bitwise LLR can be computed as:

$$\begin{aligned} L(b_i \mid Y_1^n) &= \max_{\{u',u\}\Rightarrow b_i=+1}(\alpha_{i-1}(u') + \gamma_i(u', u) + \beta_i(u)) \\ &\quad - \max_{\{u',u\}\Rightarrow b_i=-1}(\alpha_{i-1}(u') + \gamma_i(u', u) + \beta_i(u)) \\ &= \max_{\{u',u\}\Rightarrow b_i=+1}\sigma_{i1}(u', u) - \max_{\{u',u\}\Rightarrow b_i=-1}\sigma_{i0}(u', u) \end{aligned} \tag{2.13}$$

2.4 Software Watermarking

Digital watermarking has been traditionally used to provide copyright protection for different kinds of digital objects. In the copyright protection scenario, a distributor embeds the watermark into the digital object, so its ownership can be proved later. Software watermarking is the term used when the digital object is a software application. Software watermarking has been used to detect software piracy (i.e. the illegal copying and resale of software applications). In addition, software watermarks have also been used in other scenarios such as tamperproofing or obfuscation [21].

According to [23] there are three parameters that essentially define the characteristics and security of a software watermarking scheme: (1) *the data rate* expresses the quantity of hidden data that can be embedded within the digital object; (2) *the stealth* expresses how imperceptible the embedded data is to an observer; and (3) *the resilience* expresses the hidden message's degree of immunity to attacks performed by

an adversary. All watermarks exhibit a trade-off between these three parameters and the related cost.

2.4.1 Classification of Software Watermarks

Software watermarks are usually classified in two types:

- **Static watermarks.** The static watermark is embedded in the executable code of the program. The main drawback of static watermarks is that they can be detected even without running the program and thus, they are susceptible to attack by anyone of reasonable skills in software analysis. One of the most robust static watermarking techniques is presented in [111]. In that proposal, Venkatesan *et al.* treat the program as a control flow graph, in which a watermark graph is added to form the marked program.

- **Dynamic watermarks**. The watermark depends on conditions during the execution of the program. These conditions can be related with input data, user-interaction, a packet from network, a special file, the program state etc. This makes dynamic watermarks much more difficult to detect because in general the application must be run several times to detect the watermark. As dynamic watermarks are relatively new, there are still few published proposals of this kind [23, 87]. In the literature we can find three types of dynamic watermarks:

  - *Easter egg watermarks*, in which the application performs an action that is immediately perceptible for the user when a special input sequence is entered.
  - *Execution trace watermarks*, in which the watermark is embedded within the program trace (either instructions or addresses).
  - *Data structure watermarks*, in which the watermark is embedded within the state of the program (global, heap, stack data, etc.).

2.4.2 Threat Model for Software Copyright Protection

The objective of an illegal software redistributor is to make the watermark invalid without changing the behavior of the program. In this sense, three main attacks can be performed:

- **Subtraction attacks:** an attacker that knows the location of the watermark can try to delete it from the code, in the hope that the program after the extraction will be still useful.

- **Distortive attacks:** an attacker without knowledge about the location of the watermark can apply transformations that uniformly distort the code trying to make the watermark unrecognizable.

- **Additive attacks:** an attacker adds its own mark, in the hope that it will be impossible to detect that the real watermark precedes the new fake one.

Most distortive attacks are based on semantic preserving program transformations, that is, transformations that preserve the semantics of the program, but they modify its appearance. Some examples of classical semantic preserving program transformations are obfuscation, translation and optimization (compilation, decompilation or binary translation).

Regarding the strength of both types of software watermarks, in general, static watermarks are simpler than the dynamic ones, but also weaker against attacks [23, 21]. Static watermarks are usually easy to distort by using any semantic preserving program transformation. On the other hand, most published dynamic watermarking schemes are resilient to some of these transformations when applied individually, but not to combined attacks of some of them. For this reason, most watermarking schemes are designed to make it difficult to locate and change the watermark when semantic preserving program transformations are used.

2.4.3 Dynamic Graph Watermarking

The CT algorithm (also called *Dynamic Graph Watermarking* [23]) is based on embedding watermarks within the topology of graphs built dynamically in memory during the execution of a program. The structure embedded is a *graph-watermark* (G). The *graph-watermark* contains in its topology a representation of a number N, which is the product of two large primes p and q. A program called *recognizer* or R can retrieve this graph from memory. Then, N can be retrieved from G, and finally, the author can prove that she has embedded the corresponding graph into the code because she knows p and q.

Graphs are a suitable mechanism to embed marks because as it is known [23], the analysis of large graphs involves significant complexity and requires an important computational effort. Furthermore, attacks based on semantic modifications of the source code are useless because they do not alter the execution of the marked code, and thus, the watermark can be recovered by means of the analysis of the memory during the execution.

Figure 2.6 illustrates the steps of this mechanism. The algorithm starts selecting two large primes (p and q) and calculating $N = p \times q$. Then, the algorithm continues as follows:

1. Embedding N into the topology of a given graph G.
2. Creating the code W which generates G.
3. Embedding W in the original code O to generate a O_0, which given an input I, the recognizer R is able to extract W (and hence N).
4. Using tamperproofing to avoid W being removed (generating O_1).
5. Using obfuscation to difficult analysis (generating O_2). In this case, the recognizer R and the watermark code W become R' and W' respectively because of obfuscation.
6. Extracting the recognizer R' and distributing the marked code O_3.
7. After distribution, an attacker can generate O_4 distorting the code O_3 to make the watermark invalid.
8. The author of the code can prove her authorship by applying the recognizer R' to O_4 using the special input I. This will generate the graph G in memory, so N can be found. As N is not a random number but it has been chosen deliberately as the product of two large primes, then the author demonstrates authorship just factoring N (publishing p and q).

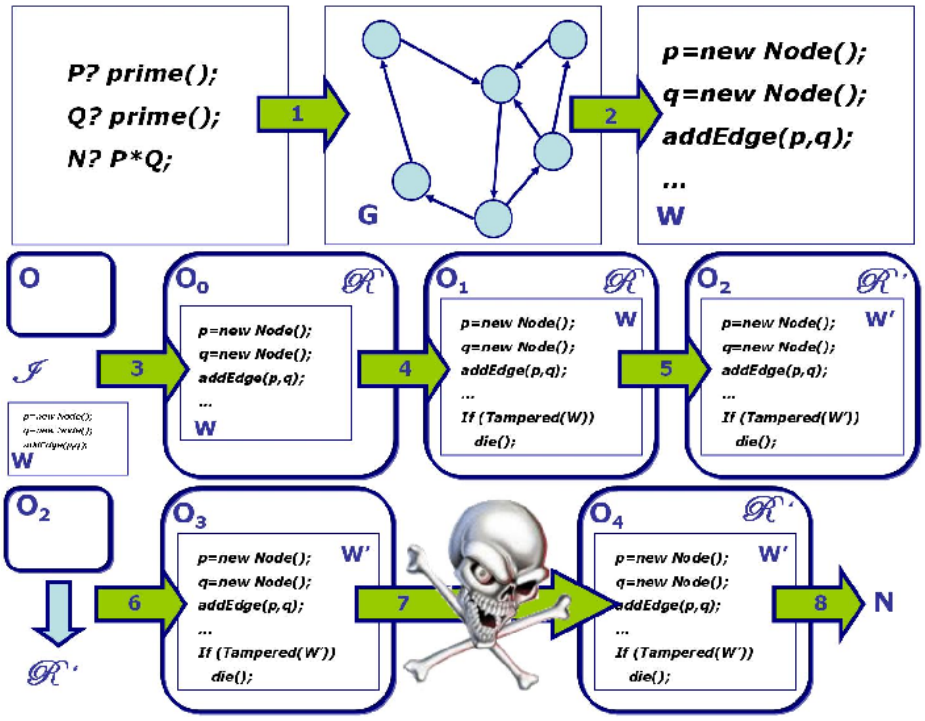


Figure 2.6: Scheme of Dynamic Graph Watermarking

One of the main difficulties of this algorithm is the embedding process of the mark within a graph. Collberg *et al.* mention in [23] some possible ways to do so. From these, we summarize here the Radix-k encoding. In Radix-k, the number used as watermark is embedded by means of a circular linked list. In fact, every number can be encoded as $n = \sum_{i=0}^{k-2} a_i k^i$. So, the base-$k$ digit is encoded by the length of the list and an extra pointer, which points to the first node. Every node encodes an a_i by pointers: if the pointer is null, then $a_i = 0$; if the pointer points itself, then $a_i = 1$; if it points the next element, then $a_i = 2$, and so on. Equation 2.14 and Figure 2.7 show an example of Radix-6 encoding, which codifies the number $n = 4453$ with these coefficients $a = \{a_0, a_1, a_2, a_3, a_4\} = \{1, 4, 3, 2, 3\}$.

$$n = 3 \times 6^4 + 2 \times 6^3 + 3 \times 6^2 + 4 \times 6^1 + 1 \times 6^0 = 4453 = 61 \times 73 \tag{2.14}$$

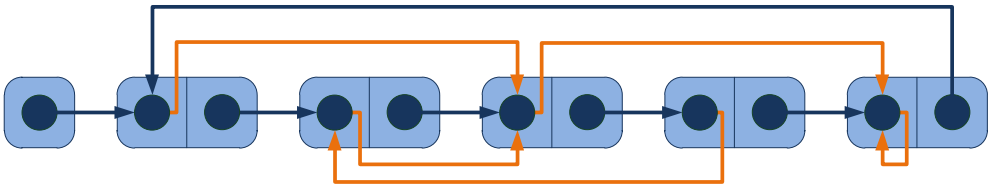


Figure 2.7: Example of embedding n=4453 by means of Radix-k

Enumeration Encoding: This method is based on the possibility of enumerating and indexing graphs [54]. The idea is codifying N using the index of the graph in this enumeration.

2.4.4 Self-Validating Branch-Based Software Watermarking by Myles *et* al.

The algorithm presented in [87] fulfils the watermarking requirements and it also contributes with fingerprinting characteristics to the system. In other words, this algorithm can embed imperceptible authorship information in a piece of code and besides, it adds mechanisms to identify the owner of a specific copy of this piece of code. The Self-Validating Branch-Based Software Watermarking algorithm, as its name indicates, is based in the use of branch function to generate, in runtime, the appropriate fingerprint codeword. As a collateral effect of the use of this type of functions, tamper detection can be incorporated into this algorithm to detect and to disappoint attacks. This algorithm has twice different processes. The first process (equation 2.15) is the embedding operation (`embed`). This function requires 4 input parameters: the piece of code to be marked (P), Authorship mark (AM) and two secret keys, KEY_{AM} and KEY_{FM}. Note that, while the KEY_{AM} is the same for all copies, each copy will have a

different unique KEY_{FM}. The aim of this function is to add the authorship mark and the fingerprint generating code into the program (P), yielding a new piece of code or program (P') and a fingerprinting mark (FM) which is generated from KEY_{FM} and the program execution trace. The second process (`recognise`) the function which can retrieve the authorship mark and the fingerprinting mark from a marked piece of code or program, the key_{AM} and key_{FM} (equation 2.16). Note that this algorithm has a blind recogniser, that is, marks can be retrieved without the original piece of code (P), only the marked program P' and the respective keys key_{AM} and key_{FM} are needed.

$$\texttt{embed}\left(P, AM, key_{AM}, key_{FM}\right) \longrightarrow P', FM \tag{2.15}$$

$$\texttt{recognise}\left(P', key_{AM}, key_{FM}\right) \longrightarrow AM, FM \tag{2.16}$$

The branch functions [76] were created to difficult static disassembly of native executables as a basis of an obfuscation technique. Its operation is conceptually easy (see figure 2.8): some determined jump instructions of the code are replaced by calls to the same function which will redirect these calls to the correct point in the code. Myles *e*t al. defined a special type of branch functions that is named *fingerprinting branch function* (FBF). In addition to normal branch functions behaviour, this kind of function modifies the k_i value every time that this function is called. This value is used to obtain the destination of the branch in each iteration. Moreover, in the last iteration, k_n will take as value the fingerprinting mark FM. The aims of FBF can be schematized as follow: Verifying code integrity by obtaining, with a digest function, the value v_i which will be used in the k_i calculation; to generate the new k_i value with one way function which depends on the values of k_{i-1} and v_i; transferring program execution to the original branch target using the value of k_i; tamper detection; authorship mark is incorporated to prove ownership. As example of k_i calculation, Myles *e*t al., propose $k_i = SHA1[(k_{i-1} \oplus AM) \| v_i]$. Note that the integrity checks, as a tamper detection mechanisms, are capable of detecting if a program has been subjected to semantics-preserving transformation or even if a debugger is presented. Figure 2.9 shows a simplified schema of a tamper detection mechanism. For example, if malicious host is analysing the agent execution (inserting, for instance, breakpoints in a debugger), the checksum over a block of code will be different from the original checksum and the branch function will not be able to find the program target for this adulterated checksum.

The embedding process can be divided into three steps. The first step lies in running the program to be marked using as input the authorship key (key_{AM}) to obtain the trace of the program. In the second step of the algorithm the branches in each function $f \in F$ (where F is the set of all functions identified by trace) are substituted by calls to the FBF. The values for k_i and the mapping between these values and the pointers to the next respective steps are generated. Finally, the resulting structure of

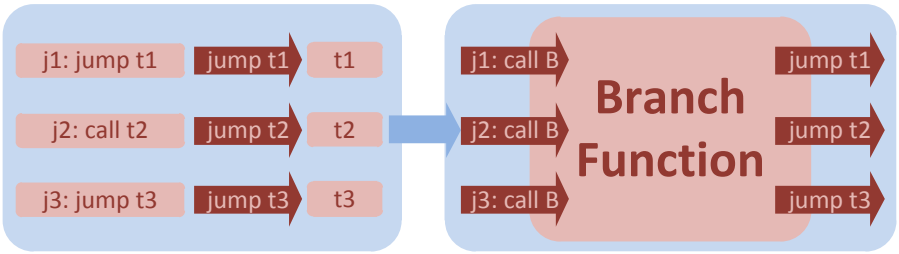


Figure 2.8: Some instructions (jump or call for instance) are converted to calls to a branch function. The next step in the program execution will be managed by this branch function.

the mapping is injected into the code. Note that this mapping is essential to the correct behaviour of the application.

When the program is executed taking the pertinent keys as input, the set of the functions marked by fingerprinting process and the FBF are identified. If the one way function used is known, the supposed author can demonstrate his authorship by supplying his authorship mark and comparing with the obtained mark. In the same way, as a result of the last call to FBF, the fingerprint mark is obtained.

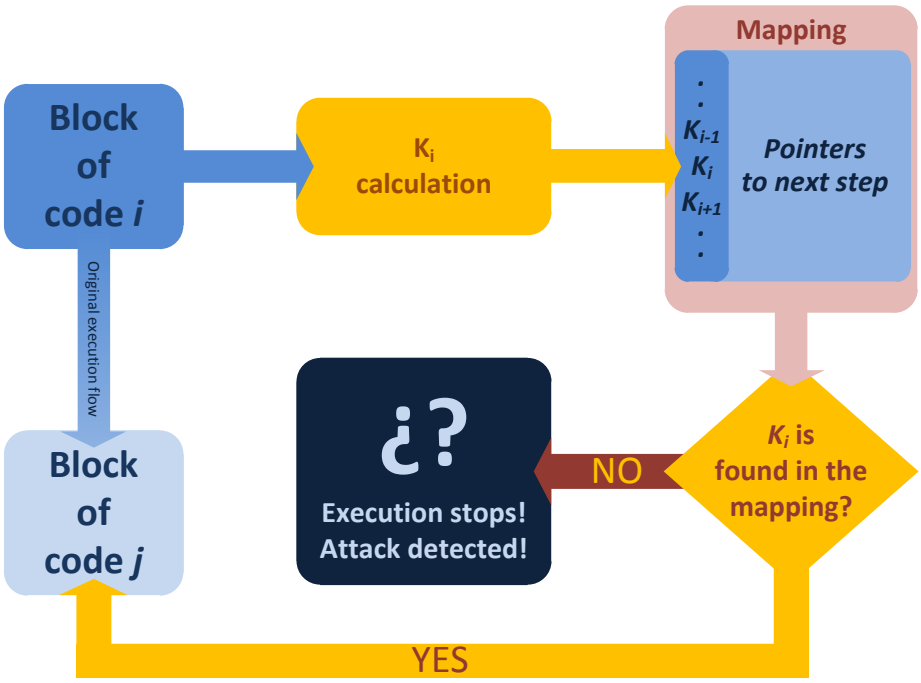


Figure 2.9: Tamper detection mechanism implemented with checksums and branch functions.

2.5 Mobile agents

Mobile agents are software entities that consist of code, data and state, and that can migrate from host to host performing actions on behalf of a user. For instance, mobile agents can be used to support wireless terminals, which usually are resource-constrained. In fact, mobile agents are especially useful to perform functions automatically in almost all electronic services, like distributed computing, e-commerce or data mining. Figure 2.10 shows a possible mobile agent scenario.

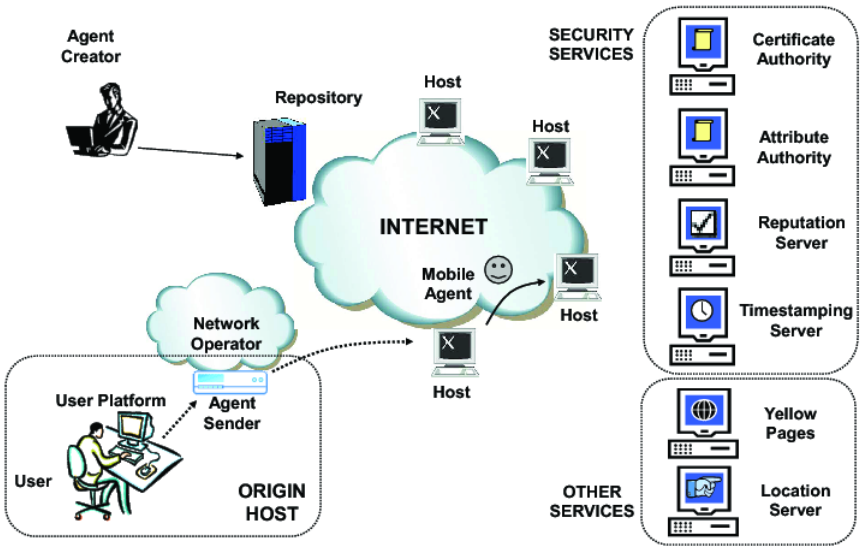


Figure 2.10: Mobile Agent Scenario

In this scenario, a user wants to perform some tasks using mobile agents. For instance, let us imagine that the user wants to arrange a meeting in which invitees are spread all over the country in different local offices. The date and place of the meeting may depend on several different parameters such as room availability, room capacity, available commodities (projector, blackboard, etc) and, of course, on the invitees' agendas. To do so, the user will use a mobile agent that will migrate to the host of each invitee to consult her agenda, room availability in the local office and so on. Finally, the mobile agent will decide the place and date of the meeting, and it will introduce a new entry in the agenda of each invitee. The user might send the agent by herself, or she might delegate the sending process to an Agent Sender, which is a principal with capabilities to dispatch the mobile agent to the network. Usually, this part of the mobile-agent infrastructures are called "Origin Host" and it is considered trustworthy.

On the contrary, executing hosts cannot be considered trusted entities. For instance, these executing hosts might be competitors. Before sending the agent, the user can have information about the trustworthiness of the hosts, for instance obtained in previous transactions, or consulted to external reputation servers. Depending on the

degree of trustworthiness of these hosts, the user can decide if protection techniques are needed or not, or it simply can avoid sending the agent to them. There are several reasons that can lead a host to become malicious and attack a mobile agent to obtain a certain profit. In the previous example, one of the invitees could try to manipulate the agent to impose the meeting to be in her local office because she is averse to travel, or simply to avoid spending money in a flight. Even, she can try to impose a date in which an adversary of another local office cannot assist to the meeting to damage the reputation of this adversary.

According to reasons exposed before, two entities must be considered to study the security weaknesses of the scenario: the mobile agent and the executing host (or simply host). Protection is necessary when trustworthy relationships between these entities cannot be assured. So these are the main cases that can be found [17, 93, 63]:

- **Host protection**: the system administrator's duty is protecting the hosts from attacks that try to exploit the weaknesses of the execution platform. The attacks performed by external entities (e.g. other hosts) are out of the scope of this article because they can appear irrespective of the use of mobile agents. However, there are new threats that must be taken into account. The hosts must be protected against the attacks that an agent can perform while it is executing its code. A malicious mobile agent can try to eavesdrop or manipulate any kind of available data, or even it can perform a denial of service attack to make the platform break down. Continuing with the example of the arrangement of a meeting, a malicious mobile agent could try to delete entries that should not have permission to access in the agenda of a user. Fortunately, most of these attacks can be detected or avoided by using a proper access control and sandboxing techniques, which are techniques that control the execution environment. In addition, some other proposals try to determine if the agent's code is malicious before executing it [37, 90].

- **Communications security**: the agent can receive multiple attacks from any external entity while it is migrating from host to host. Typical attacks are eavesdropping or manipulation of data. Data eavesdropping attacks can be avoided by using encryption techniques, so any confidential information should be encrypted. Data manipulation attacks can be detected by using digital signature [68]. This includes any part of the mobile agent (the code, the itinerary, etc). In fact, before sending the agent the origin host signs the mobile agent to assure that it will not be modified during its journey by any host. So then, the hosts are sure that the agent's code has not been manipulated by any external entity (this includes any other previous host in the itinerary). In both cases, data eavesdropping and manipulation, the use of well-known cryptographic protocols is nec-

essary. However, these cryptographic techniques are subject to the existence of mechanisms to create and exchange encryption/signature keys between hosts. The use of a Public Key Infrastructure (PKI) is a useful and standard way to provide these services. However, any other mechanism/infrastructure that makes possible the creation and exchange of encryption/signature keys could be used.

- **Agent protection**: while the agent is inside an execution platform, it can receive attacks from other agents or from the platform itself. The attacks between agents inside the same platform can be solved by using separate execution environments. However, it is not so easy to protect the agents from the attacks of the execution platform. This is considered the most difficult security problem to solve in mobile agent systems for most of the authors [36, 61]. There are many reasons that can lead a host to attack the agents that is executing. The host can try to obtain an economical benefit or a favorable execution, or just it can try to damage the reputation of another principal. Following the previous example, a rival host could try to manipulate the agent to impose the meeting in another local office, but in a room without projector, so the organizer of that local office will make a bad impression. Notice that while it is possible to assure the agent's privacy and integrity during its journey, It is difficult to prevent or detect eavesdropping and manipulation attacks performed by the host during the execution. The host has complete control of the execution and hence it can read or modify any part of the mobile agent: the code, the execution flow, the state, the itinerary, the communications, or even the results. The agent cannot hold a decryption key because the host could read or modify it. This is the reason why there is not a published solution that protects the mobile agents completely from the attacks of an executing host. This kind of attacks is also known as the problem of the "*malicious hosts*".

As the attacks performed by malicious hosts are considered the most difficult to face within the mobile agent scenario regarding security [16], the research in this area is centred on it. There are two main attacks of this kind: (1) eavesdropping attacks, in which the host tries to extract information from the execution of the agent. The system must provide execution privacy to face these attacks, but this security service is difficult because eavesdropping attacks cannot be detected, only avoided; and (2) manipulation attacks, in which the executing host tries to modify the proper execution of the agent. Providing execution integrity is also a quite difficult security service because the executing hosts have complete control over the agent's execution.

The literature about countermeasures for malicious host attacks can be divided in two kinds of approaches: attack avoidance and attack detection approaches. Regarding attack avoidance approaches, they try to avoid attacks before they happen. Some

authors introduced the idea of a tamper-proof hardware subsystem [119, 79] where agents can be executed in a secure way, but this forces each host to buy this hardware. Hohl presents obfuscation [58] as a mechanism to assure the execution integrity during a period of time, but this time depends on the capacity of analyzing the code of the malicious host. The use of encrypted programs [99] is proposed as the only way to give execution privacy and integrity to mobile code. The difficulty here is to find functions that can be executed in an encrypted way. Published attack avoidance techniques are difficult to implement or computationally expensive. For this reason, we consider attack detection techniques more promising because they are usually easier to implement. The objective of attack detection approaches is detecting manipulation attacks. In [83], the authors introduce the idea of replication and voting, but this proposal can only be used as an attack detection approach if the hosts in the same stage are independent. In [112], Vigna introduces the idea of the cryptographic traces, which are logs of the operations performed by the agent. The operations of the agent can be categorized in white statements, which modify the agent's state due to internal variable values; and black statements, which alter the agent's state due to external variables. These traces contain the changes performed to internal variables as a consequence of black statements. A re-execution of the agent can be performed with these traces.

Part II

Contributions related to fingerprinting codes and schemes

CHAPTER 3

IMPROVEMENTS OF EXISTENT CONVOLUTIONAL-LIKE FINGERPRINTING CODES

3.1 Introduction

The distribution and playback of digital images and other multimedia products is easy due to the digital nature of the content. Achieving satisfactory copyright protection has become a challenging problem for the research community. Encrypting the data only offers protection as long as the data remains encrypted, since once an authorized but fraudulent user decrypts it, nothing stops him from redistributing the data without having to worry about being caught.

The concept of fingerprinting was introduced by Wagner in [116] as a method to protect intellectual property in multimedia contents. The fingerprinting technique consists in making the copies of a digital object unique by embedding a different set of marks in each copy. Having unique copies of an object clearly rules out plain redistribution, but still a coalition of dishonest users can collude. A collusion attack consist in comparing the copies of the coalition members and by changing the marks where their copies differ, they create a pirate copy that tries to disguise their identities. In this situation they can also frame an innocent user. Thus, the fingerprinting problem consists in finding, for each copy of the object, the right set of marks that help to prevent collusion attacks.

The construction of collusion secure codes was first addressed in [12]. In that paper, Boneh and Shaw obtain ($c > 1$)-secure codes, which are capable of identifying a

guilty user in a coalition of at most c users with a probability ϵ of failing to do so. The construction composes an inner binary code with an outer random code. Therefore, the identification algorithm involves the decoding of a random code, that is known to be a NP-hard problem [6]. Moreover, the length of the code is considerably large for small error probabilities and a large number of users.

To reduce the decoding complexity, Barg, Blakley and Kabatiansky in [6] used algebraic-geometric codes to construct fingerprinting codes. In this way, their system reduces the decoding complexity to $O(poly(n))$ for a code length n and only 2 traitors. In [38], Fernandez and Soriano constructed a 2-secure fingerprinting code by concatenating an inner $(2,2)$-separating codes with an outer IPP code (a code with the Identifiable Parent Property), and also with decoding complexity $O(poly(n))$.

The Collusion Secure Convolutional Fingerprinting Information Codes presented in [122] have shorter information encoding length and achieve optimal traitor searching in scenarios with a large number of buyers. Unfortunately, these codes suffer from an important drawback in the form of false positives, in other words, an innocent user can be tagged as guilty with very high probability. This codes where extended by Zhang *et al.* in the paper [121] using an turbo code instead of a convolutional code. In a practical implementation of these codes, the turbo code must have some restrictions, which the authors did not take into account, to obtain the desired performance.

In this chapter, these two proposals are analysed in depth and some improvements are proposed. More precisely, in section 3.4 we analyse in depth the work in [122] and quantify the probability of false positives. Moreover, in section 3.5 turbo fingerprinting codes are analysed. This analysis shows that, in order to obtain the desired performance, the turbo code must have some restrictions in a practical implementation of these codes. The authors did not take into account this restrictions.

The chapter is organized as follows. In Section 3.2 we provide some definitions on fingerprinting and error correcting codes. Section 3.3 presents the well known Boneh-Shaw fingerprinting codes. Section 3.4 discusses the Collusion Secure Convolutional Fingerprinting Information Codes presented by Zhu *e*t al. and carefully explains the encoding and decoding mechanisms. Furthermore we deal with the false positive problem and give a bound on the false positive probability and some guidelines for the correct construction of this family of codes with low false positive probability are presented. Section 3.5 discusses the turbo fingerprinting codes presented by Zhang *e*t al. and carefully explains the encoding and decoding mechanisms. Moreover, new considerations about Turbo Fingerprinting Codes (TFC) as well as the errors that can be produced as a consequence of not taking into account these considerations are explained and justified. In the same way a numerical example of this problem is given. Furthermore, this section proposes two improvements to the performance of the TFC using the likelihood provided by the turbo decoder. Finally, some conclusions are

given in Section 3.6.

3.2 Definition

We begin by defining some concepts which will be needed throughout this chapter.

Definition 3.1 (Error Correcting Code)**.** *A set C of N words of length L over an alphabet of p letters is said to be an $(L,N,D)_p$-Error-Correcting Code or in short, an $(L,N,D)_p$-ECC, if the Hamming distance*[1] *between every pair of words in C is at least D.*

Other important definitions needed below are:

Definition 3.2 (Convolutional Code)**.** *A convolutional code is a type of error-correcting code in which each k-bit information symbol (each k-bit string) to be encoded is transformed into an n-bit symbol, where k/n is the code rate ($n \geq k$) and the transformation is a function of the last m information symbols, where m is the constraint length of the code. Or more formally [39], an (n,k) convolutional encoder over a finite field F is a k-input n-output constant linear causal finite-state sequential circuit. And a rate k/n convolutional code C over F is the set of outputs of the sequential circuit.*

An important concept in fingerprinting environments is the *Marking Assumption.*

Definition 3.3 (Codebook [12])**.** *A set $\Gamma = \{W^{(1)}, W^{(2)}, \cdots, W^{(n)}\} \subseteq \Sigma^l$, where Σ^l will denote some alphabet of size s, will be called an $l(l,n)$-code. The codeword $w^{(i)}$ will be assigned to user u_i, for $1 \leq i \leq n$. We refer to the set of words in Γ as the **codebook***

Definition 3.4 (Undetectable Position)**.** *Let $\Gamma = \{W^{(1)}, W^{(2)}, \cdots, W^{(n)}\}$ is an (l,n)-code and $C = \{u_1, u_2, \cdots, u_c\}$ is a coalition of c-traitors. Let position i be **undetectable** for C, i.e. the words assigned to users in C match in i'th position, that is $w_i^{(u_1)} = \cdots = w_i^{(u_c)}$.*

Definition 3.5 (Feasible set)**.** *Let $\Gamma = \{W^{(1)}, W^{(2)}, \cdots, W^{(n)}\}$ is an (l,n)-code and $C = \{u_1, u_2, \cdots, u_c\}$ is a coalition of c-traitors. We define the **feasible set** Γ of C as*

$$\Gamma(C) = \{x = (x_1, \cdots, x_l) \in \Sigma^l \mid x_j \in w_j, 1 \leq j \leq l\}$$

where

$$w_j = \begin{cases} \{w_j^{(u_1)}\} & w_j^{(u_1)} = \cdots = w_j^{(u_c)} \\ \{w_j^{(u_i)} \mid 1 \leq i \leq c\} \cup \{?\} & otherwise \end{cases}$$

where ? denotes an erased position.

[1] The Hamming distance $d(y;x)$ [53] between two sequences of equal length can be defined as the number of positions in which the two sequences differ.

Now we are in position to define the *Marking Assumption* that establishes the rules that the attacking coalition are subjected to. This definition sets the work environment of many actual fingerprinting systems.

Definition 3.6 (Marking Assumption)**.** *Let* $\Gamma = \{W^{(1)}, W^{(2)}, \cdots, W^{(n)}\}$ *be an (l,n)-code,* $C = \{u_1, u_2, \cdots, u_c\}$ *a coalition of c-traitors and* $\Gamma(C)$ *the feasible set of C. The coalition C is only capable of creating an object whose fingerprinting lies in* $\Gamma(C)$.

The main idea of this definition is that a coalition of *c*-traitors can not detect the positions in the document in which their mark holds the same value. Many of the fingerprinting schemes in the literature base their tracing algorithms in trying to estimated the positions that are changed by the attackers.

As example, a (3,4)-code can be defined as:

Positions:	**1**	**2**	**3**
A:	1	1	1
B:	0	1	1
C:	0	0	1
D:	0	0	0

Suppose that users B and C collude, for this collusion the positions 1 and 3 will be ***undetectable positions***. And the ***Feasible set*** for this collusion will be:

0	1	1
0	?	1
0	0	1

And, taking into account the marking assumption, the coluders are only capable of creating an object whose fingerprinting lies in this set of words. Note that, as well as putting one of their value in the position 2, they can erase this position which is represented by ?.

3.3 Boneh-Shaw fingerprinting model

In 1995, Dan Boneh and James Shaw presented in [12] a seminal paper about the collusion secure fingerprinting problem. First of all, we need to define what a fingerprinting scheme is.

Definition 3.7 (Fingerprinting scheme [12])**.** *A* (l,n)*-fingerprinting scheme is a function* $\Gamma(u,r)$ *which maps a user identifier* $1 \geq u \geq n$ *and a string of random bits* $r \in \{0,1\}^*$ *to a codeword* Σ^l*. The random string r is the set of random bits used by the distributor and kept hidden from the user. We denote a fingerprinting scheme by* Γ_r.

3.3.1 n-secure codes

We now define n-secure codes, see [12] for a more detailed description.

Definition 3.8 (c-secure code with ϵ-error)**.** *A fingerprinting scheme Γ_r is a c-secure code with ϵ-error if there exists a tracing algorithm A which from a word x, that has been generated (under the Marking Assumption) by a coalition C of at most c users, satisfies the following condition $Pr[A(x) \in C] > 1-\epsilon$ where the probability is taken over random choices made by the coalition.*

Now, we define the code and its decoding algorithm:

1. Construct an n-secure (l,n)-code with length $l = n^{O(1)}$.

2. Construct an $\Gamma_0(n,d)$-fingerprinting scheme by replicating each column of an (l,n)-code d times. For example, suppose a $(3,4)$-code {111,011,001,000}. We can construct a $\Gamma_0(4,3)$ for four users A,B,C and D as follows:

$$\begin{aligned} A&: \quad 111111111 \\ B&: \quad 000111111 \\ C&: \quad 000000111 \\ D&: \quad 000000000 \end{aligned}$$

3. When the code has been defined, the next step is to define the appropriate decoding algorithm. For instance

   Algorithm 3.1. *From [12], given $x \in \{0,1\}^l$, find a subset of the coalition that produced x. We denote by B_m is the set of all bit positions in which the column m is replicated, $R_m = B_{m-1} \cup B_m$ and weight denotes the number of bits that are set to 1.*

   a) If weight $(x \mid B_1) > 0$ then output "User 1 is guilty"

   b) If weight $(x \mid B_{n-1}) < d$ then output "User n is guilty"

   c) For all $s = 2$ to $n-1$ do:

   Let $k = weight\,(x \mid R_s)$. if

$$weight(x \mid B_{s-1}) < \frac{k}{2} - \sqrt{\frac{k}{2}\log\frac{2n}{\epsilon}}$$

   then output "User s is guilty"

Finally, the only thing left to do is to find a relationship between the error ϵ and the replication factor d. This relation is given in the following theorem,

Theorem 3.1. *For $n \geq 3$ and $\epsilon > 0$ let $d = 2n^2 \log(2n/\epsilon)$. The fingerprinting scheme $\Gamma_0(n,d)$ is n-secure with ϵ-error and has length $d(n-1) = O(n^3 \log(n/\epsilon))$.*

3.3.2 Logarithmic Length c-Secure Codes

The construction of Boneh and Shaw n-secure with ϵ-error is impractical for a medium and large number of user because the length of codewords increases as $O(n^3 \log(n/\epsilon))$. To achieve shorter codes, Boneh and Shaw apply the ideas of [18] to construct c-secure (n, l)-codes of length $l = c^{O(1)} \log(n)$. The basic idea is to use the n-secure code as the alphabet which is used by an $(L, N, D)_p$-error-correcting code. As a result of this composition, Boneh and Shaw obtained the following result. The proof of this theorem can be found in [12].

Theorem 3.2. *Given integers N,c, and $\epsilon > 0$ set $n = 2c$, $L = 2c\log(2N/\epsilon$, and $d = 2n^2 \log(4nL/\epsilon)$. Then, $\Gamma'(L, N, n, d)$ is a code which is c-secure with ϵ-error. The code contains N words and has length $l = O(Ldn) = O(c^4 \log(N/\epsilon)\log(1/\epsilon))$*

Thus the code $\Gamma'(L, N, n, d)$ is made up of L copies of $\Gamma_0(n, d)$. Each copy is called a *component* of $\Gamma'(L, N, n, d)$. The codewords of component codes will be kept hidden from the users. Finally, the codewords of $\Gamma'(L, N, n, d)$ are randomly permuted by π before the distributor embeds the codeword of the user u_i in an object, that is to say, user u_i's copy of the object will be fingerprinted using the word $\pi w^{(i)}$. To guarantee the security of this scheme, the permutation π must be kept hidden from the users in order to hide the information of which mark in the object encodes which bit in the code.

3.4 Improvement of Collusion Secure Convolutional Fingerprinting Information Codes

This sections presents, analyzes and improves the false positive problem detected in Convolutional Fingerprinting Information Codes presented by Zhu *e*t al. The codes are a concatenation of a convolutional code and a Boneh-Shaw code. In their paper Zhu *e*t al. present a code construction that is not necessarily c-secure with ϵ-error because in their modified detection algorithm the standard Viterbi error probability analysis cannot be directly applied. In this case we show that false positives appear, that is to say, the problem of accusing an innocent and legal user of an illegal redistribution, and give a bound on the probability of false positives as well as justify it analytically and by simulations. Moreover, some guidelines for a correct design of this family of codes is also given.

3.4.1 Collusion Secure Convolutional Fingerprinting Information Codes

In [122], Yan Zhu *e*t al. presented a fingerprinting scheme with the aim of obtaining a c-secure with ϵ-error fingerprinting schemes for N-users that improve BS in run-time and storage capacity. Mainly, their scheme composes an inner Boneh-Shaw code with an outer Convolutional code. In the decoding process, they propose a modified Viterbi algorithm that works with more than one symbol in each step or 'Optional Code Subset' as the authors term it. In the next subsections we explain in detail the original workflow of this scheme.

3.4.1.1 Convolutional fingerprinting encoding

The fingerprinting code presented in [122] has a two-layer concatenated structure (the Convolutional Error-Correcting Layer and the Fingerprinting Layer) and we denote it as $\Phi(L,N,l,n)$-Fingerprinting code where N is the number of users, L is the convolutional code length and l and n are the parameters of an $\Gamma(l,n)$ code. The encoding process of a $\Phi(L,N,l,n)$-fingerprinting code consists in the following 3 steps:

1. A codeword $m^{(u_i)}$ of length N is randomly assigned to each user u_i.
2. **Convolutional error-correcting layer encoding:** The codeword $m^{(u_i)}$, divided into blocks of k_0 bits, is used as input to a (n_0,k_0,m_0)-Convolutional encoder
3. **Fingerprinting layer encoding:** Each group of n_0 output bits is mapped to a $\Gamma(l,n)$-code.

To construct a $\Phi(L,N,l,n)$-code we need two codes. The first code must be a c-secure code with ϵ-error. For this purpose, the authors use the Boneh-Shaw $\Gamma_0(n,d)$ code [12].

As it has been discussed before, this code consist of all columns $(c_1,c_2,\cdots,c_{n-1})$ each duplicated d times. The value of c_i can be expressed as:

$$c_i = \begin{cases} 1 & if input\ x \geq i \\ 0 & otherwise \end{cases}$$

For example, when the input is 1 all c_i take the binary value 1, and when the input is n all c_i take the binary value 0. The value of d depends on ϵ as is shown in Theorem 3.1.

The other code that we need for an implementation of the Convolutional Fingerprinting Scheme is a $\Im(n_0,k_0,m_0)$-Convolutional code. The only requirement that the code $\Im(n_0,k_0,m_0)$ must satisfy is that $n \geq 2^{k_0 m_0}$, in other words, the number of symbols in the alphabet of the code $\Im(n_0,k_0,m_0)$ must not be larger than the number of symbols in the alphabet of code $\Gamma_0(n,d)$.

Algorithm 3.2 presents the encoding process. Finally, the result of this process must be randomly permuted by a permutation denoted by π. The result of this permutation is embedded into the original document to obtain a copy of this document customized for user u_i.

Algorithm 3.2. *From [122], Fingerprinting-Information-Encoding (original-document X,message $m^{(u_i)}$, permutation π)*

Let v = Convolutional-Encoding($m^{(u_i)}$,i)

For each $1 \leq k \leq L$

$w^{(v_k)}$=*Fingerprinting-Encoding(v_k)*

Let $W^{(v)} = \pi(W^{(v_1)} \parallel W^{(v_2)} \parallel \cdots \parallel W^{(v_L)} \parallel)$

Embed $W^{(v)}$ into document X to obtain $X^{(u_i)}$

Return marked copy $X^{(u_i)}$

End

3.4.1.2 Convolutional fingerprinting decoding

The main objective of fingerprinting decoding is to obtain a tracing algorithm that can retrieve with a very low error probability a group of traitors that created an illegal document. Of course, this group of traitors should not contain an innocent user, at least with high probability.

First of all, the fingerprinting layer is decoded using algorithm 3.1. As a result of this, an 'Optional Code Set' is retrieved from each position. An 'Optional Code Set' is the set of codewords from an $(L,N,D)_p$-ECC that must collude to obtain a pirate codeword.

In this way, in [122] the Viterbi algorithm has been modified by using as input an 'Optional Code Set' from the fingerprinting layer decoder output instead of a corrupted codeword for each state. This modification is shown in algorithm 3.3. In the classic implementation of the Viterbi algorithm, the decoder receives from the channel a set $R = (r_1, r_2, \cdots, r_L)$ where each r_i is a codeword of an ECC that contains errors. Now the decoder receives, from the Fingerprinting Layer, a set $R = (r_1, r_2, \cdots, r_L)$ where each r_i is a set of all suspicious codewords of an ECC, i.e. $r_i = \{r_{i,1}, r_{i,2}, \cdots, r_{i,t}\}$ where $t \in N$.

Algorithm 3.3. *From [122], Convolutional-Fingerprinting-Decoding (suspect-document Y, original-document X)*

Let $d_{0,i} = \infty$, $sp_{0,i} = \{\}$ *for* $(1 \leq k \leq n)$ *except* $d_{0,1} = 0$

Let $W = \pi^{-1}(X)$

For each $1 \leq k \leq L$

Let $r_k =$ *Fingerprinting-Layer-Decoding* (Y_k, X_k)

For each state s_j

For each the incoming branch $e_{i,j}$

For each the element $r_{k,l} \in r_k$ $(1 \leq l \leq m)$

$c_{k,l} = D(r_{k,l} \mid e_{i,j})$

Let $sq_{i,j} = (sp_{k,i}, e_{i,j})$, $t_{i,j} = d_{k-1,i} + min_{1 \leq l \leq m} c_{k,l}$

Let $d_{k,j} = min_i(t_{i,j})$ *for exist* $e_{i,j}$

Let $sp_{k,j} = sq_{l,j}$ *for all* $d_{k,j} == t_{l,j}$

Let $d_{L,l} = min_{1 \leq j \leq n}(d_{L,j})$

Return $M(sp_{L,l})$

End

Maximum-likelihood (ML) decoding is modified to keep, for each state, the path that has minimum Hamming distance for each codeword in 'Optional Code Set'. For example, suppose that in the j-th step the fingerprinting layer retrieves as fraudulent binary codewords $r_j = \{1011, 0011\}$. Suppose that at the entry to state S_1 there exist two input path, one from state S_1 and other from S_2 labelled as $P1 = 0000$,$P2 = 1111$ and represented in the algorithm by $e_{1,1} = 0000$ and $e_{2,1} = 0000$. The next step will be to calculate the Hamming distance between the values in r_j (there are $r_{j,1} = 1011$ and $r_{j,2} = 0011$) and the values of $e_{1,1}$ and $e_{2,1}$. The paths that arrives to state S_1 in step j are stored in $sq_{i,j}$ and the associated total costs are stored in $t_{i,j}$. The next step in the algorithm is to search in the array $t_{i,j}$ the minimum value and this value is stored in $d_{k,j}$ as minimum distance in order to arrive to step j at state S_1. Finally, all path that arrive to state S_1 at step j with an associated cost equal to $d_{k,j}$ are stored in $sp_{k,j}$. The algorithm keeps as the ML path $P2$, with input codeword being 1011.

3.4.2 A new critical performance analysis

In [122] two theorems were presented. These theorems are briefly detailed in this section with examples of their drawbacks. Some desirable characteristics to be added will be explained in depth throughout the next sections.

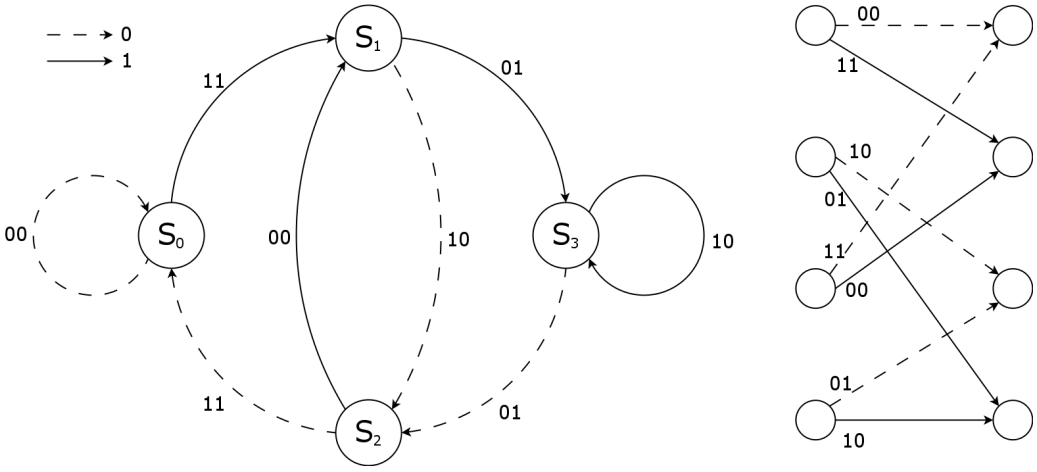


Figure 3.1: State diagram for (2,1,2) Convolutional code

Theorem 3.3. *The survivor path is the maximum likelihood path in the improved Viterbi decoding algorithm 3.3, namely, there exist a survivor path sp for all candidate paths sq ≠ sp, D(R | sp) ≥ D(R | sq).*

Let $SC(R)$ be the set of all possible combinations of suspicious codewords retrieved in each step by the Fingerprinting Layer, or more formally,

$$SC(R) = \{x = (x_1, x_2, \cdots, x_L) \mid x_j \in r_j, 1 \geq j \geq \#(r_j)\}$$

where r_j is the set of suspicious codeword retrieved by Fingerprinting Layer for the jth position in R. According to algorithm 3.3 the entry to each step consists in codewords. This idea can be seen as processing each element of $SC(R)$ with a classical Viterbi algorithm. If the fingerprinting decoding process is correct, this system will find at most c combinations all of which have zero Hamming distance with a path in the trellis diagram (3.2). The main problem is that Theorem 3.3 does not guarantee that only these c combinations are the ones to have zero Hamming distance with a path in the trellis diagram.

For example, suppose that, at random, we assign to users u_1 and u_2 the identification information $M_1 = \{0110100\}$ and $M_2 = \{1010000\}$ respectively, where the last two bits are the ending symbol for a Convolutional encoder with two memory positions. If the diagram state of [122] shown in the following figure 3.1 is used as (2,1,2)-Convolutional encoder, the codified symbols are $R^{(1)} = (00,11,01,01,00,10,11)$ and $R^{(2)} = (11,10,00,10,11,00,00)$.

The next step must be to encode $R^{(1)}$ and $R^{(2)}$ with well constructed Fingerprinting Layer, in other words, with an appropriate value of d that is robust against a coalition of two users over an alphabet of four words. After a random collusion attack of users u_1 and u_2, the resultant copy must contain a mark that the Fingerprinting Layer can

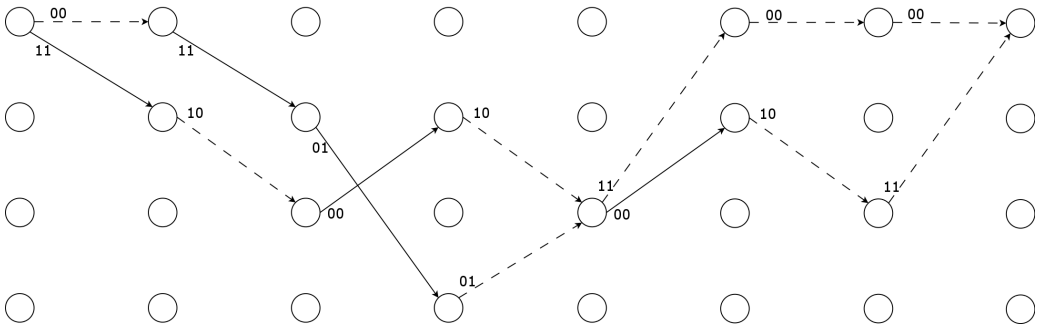


Figure 3.2: Paths in a trellis diagram corresponding to two colluders that can create false positives

retrieve

$$R' = (\{00,11\},\{11,10\},\{01,00\},\{01,10\},\{00,11\},\{10,00\},\{11,00\})$$

If algorithm 3.3 is applied to R', the trellis structure of Figure 3.2 is obtained. From this structure, the algorithm 3.3 retrieves as illegal users the ones with identification $M_1 = \{0110100\}$, $M_2 = \{1010000\}$, $M_3 = \{0110000\}$ and $M_4 = \{1010100\}$. And, in this case, users u_3 and u_4 have not done any illegal action.

Note that now is obvious that enforcing a false positive bound is necessary to design an appropriate coding scheme. Note that the problem lies in the Convolutional Error-Correcting layer not in the fingerprinting layer. In other words, we need tools to choose a correct Convolutional code for each application requirement.

Theorem 3.4. *Given integers N,c and $\epsilon > 0$, set $n = 2c$, $d = 2n^2(log(8n) + r)$, $r = (2/d_f)log(A_{d_f}/\epsilon)$, where d_f (free distance) is the minimum distance between any two codewords in the convolutional code, A_{d_f} is the number of codewords with weight d_f. Then the convolutional fingerprinting code $\Phi(L,N,l,n)$ is a code which is c-secure with ϵ-error. The code contains N codewords. Let x be a word which was produced by a coalition C of at most c users. Then algorithm 3.3 will output a codeword of C with probability at least 1-ϵ.*

The authors in [122] prove theorem 3.4, assuming the well know error probability P_e of Viterbi decoders in BSC channels (for references in error probability of Viterbi decoders in BSC channels [73, 113, 114]). The basic problem lies in the fact that a set C_1 is obtained as output of this algorithm and really, C_1 contains a codeword of C with probability at least 1-ϵ but can also contain other codewords that are not contained in C. For example, suppose that we can define $SC(R')$ with 128 possible codewords. Imagine that we take alternatively one symbol of each original codeword $R^{(3)} = (11,11,00,01,11,10,00)$. In this case, we can consider that channel error probability is close to 0,28 because 4 errors have occurred over the 14 bits transmitted (suppose that we are comparing to $R^{(1)}$).

3.4.2.1 False positive probability

It has been shown above, that a Convolutional Fingerprinting Information code exists with a non negligible false positive probability. Our approximation only takes into account the probability that a false positive can be generated when two paths have one common state in a single step i. First of all we can suppose that we have a system such as in all of the trellis states there are two input arcs and two output arcs as in Figure 3.1. On the other hand, we can suppose that there exist two colluders. Let $P(S_i^{u_1} = S_i^{u_2} = s)$ be the probability that the state of the associated path to user u_1 in step i is s and it is the same that the associated path to user u_2. This probability can be expressed formally as:

$$P(S_i^{u_1} = S_i^{u_2} = s) = P(S_{i-1}^{u_1} = s_n \cap X_i^{u_1} = x^{u_1}) + P(S_{i-1}^{u_2} = s_m \cap X_i^{u_2} = x^{u_2}) = \frac{1}{S^2 2^2} + \frac{1}{S^2 2^2} = \frac{1}{S^2 2} \tag{3.1}$$

This expression can be generalized for c colluders, a trellis with a input arcs for each state and 2^k input symbols as:

$$P(S_i^{u_x} = S_i^{u_y} = s) = \frac{a!}{(a-2)!}\binom{c}{2}\frac{1}{S^2 2^{2k}} = \frac{ac(a-1)(c-1)}{S^2 2^{2k+1}} \tag{3.2}$$

The probability that two paths stay in the same state as in step i can be expressed as:

$$P_{T_i} = \sum_{j=1}^{S} P(S_i^{u_x} = S_i^{u_y} = s_j) = S\frac{ac(a-1)(c-1)}{S^2 2^{2k+1}} = \frac{ac(a-1)(c-1)}{S 2^{2k+1}} \tag{3.3}$$

It is obvious that the probability that two paths do not stay in the same state is:

$$P_{NT_i} = 1 - P_{T_i} = 1 - \frac{ac(a-1)(c-1)}{S 2^{2k+1}} \tag{3.4}$$

And finally, taking into account the considerations presented above, the false positive probability can be approximated as:

$$P_{FP} = 1 - P_{NT_i}^{N-M-log_{a_O}S} = 1 - \left\{1 - \frac{ac(a-1)(c-1)}{S 2^{2k+1}}\right\}^{N-M-log_{a_O}S} \tag{3.5}$$

where P_{NT_i} is the probability that to paths do not stay in the same state, c is the number of colluders, M is the memory of convolutional encoder, N is the number of output symbols of the convolutional encoder, 2^k is the number of symbols of the input alphabet, a is the number of input arcs for each state, a_O is the number of output arcs for each state.

Note that a false positive can only be generated in the inner states, in other words, two paths which are crossed in the memory padding states or in the start states can not cause false positives. For example the state 0 is the last state for all paths.

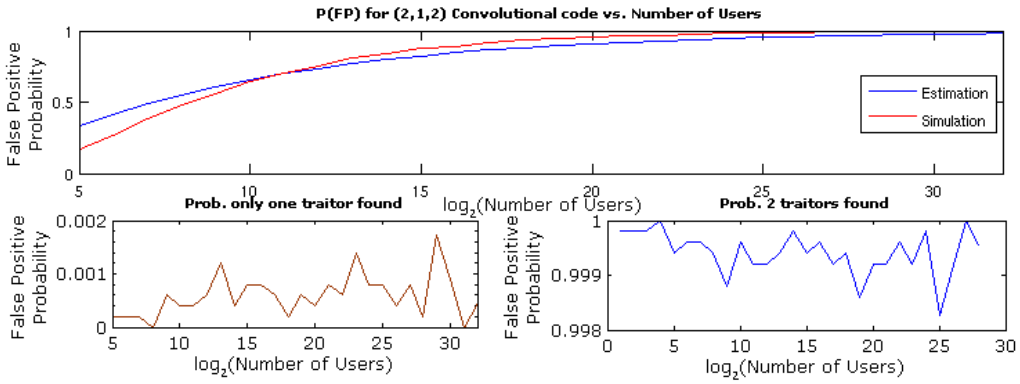


Figure 3.3: False Positive Probability *vs.* Number of users

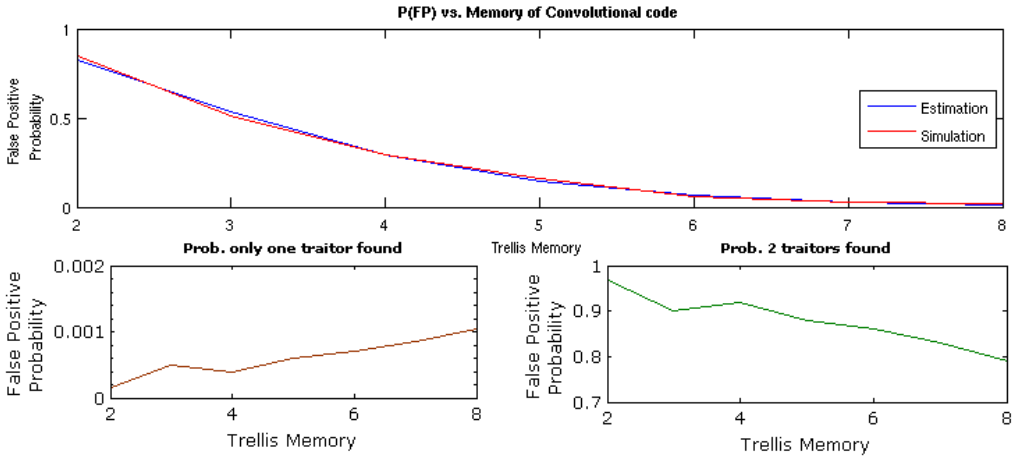


Figure 3.4: False Positive Probability *vs.* Memory of Convolutional Code

3.4.2.2 Simulation results

Figure 3.3 shows the simulation results of the behaviour of a CSCFI code with the convolutional code being the one shown in figure 3.1 after 5000 iterations for each number of users. The parameters of the BS code are $d = 331$, $\epsilon = 0.0001$ and $n = 4$. As we can imagine, when the number of users in the system increases, the number input bits increases and the false positive probability goes to 1.

On the other hand, as figure 3.4 shows, when the complexity of the trellis increases (in this simulation we use Convolutional codes from table 11.1 in [73] with ratio 1/4) the false positive probability goes to 0.

3.4.3 Guidelines for minimizing the effect of false positives

Basically, false positives can be caused by two reasons:

- **Decoding error in the Fingerprinting Layer:** This type of errors can be avoided

by a good design of this layer. If the Boneh-Shaw Fingerprinting code is used, the parameter d will be dimensioned with an appropriate error probability. Normally, the inner code will have a low number of codewords and, for this reason, it will be easy to configure the system to obtain a very low error probability in this layer. After this, we can consider that the origin of false positives is the following item.

- **Path Crossing in the Convolutional Layer:** In [122], Zhu et. al. discuss an error probability condition. This condition has to be satisfied by equation 3.5. With this two conditions in mind we can design an effective Collusion Secure Convolutional Fingerprinting Information (CSCFI) Code with ϵ-error probability. As we notice in equation 3.5, convolutional encoders that are used for the outer codes need a big complexity in their trellis structure in order to offer protection against Path Crossing. In the same way, Convolutional encoders with high ratios and a large number of symbols are more appropriate.

On the other hand, when a system with a complex convolutional encoder is designed, the coding and decoding time increases. This situation defines a relationship between the trellis complexity and the computational complexity. It will need to prioritize one of them.

3.5 New considerations about the correct design of Turbo Fingerprinting Codes

Since the introduction of turbo codes in 1993, many new applications for this family of codes have been proposed. One of the latest, in the context of digital fingerprinting, is called turbo fingerprinting codes and was proposed by Zhang *et al.*. The main idea is a new fingerprinting code composed of an outer turbo code and an inner code based on the Boneh-Shaw model. The major contribution of this section is a new analysis of this new family of codes that shows its drawbacks. These drawbacks must be considered in order to perform a correct design of a turbo fingerprinting scheme otherwise the scheme cannot retrieve the traitor users which is the main goal of digital fingerprinting scheme. Moreover, the identification of these drawbacks allows to discuss an entirely new construction of fingerprinting codes based on turbo codes.

3.5.1 Turbo Codes

Turbo codes were introduced in 1993 by Berrou, Glavieux and Thitimajashima [9], [10]. In their research, they reported extremely impressive results for a code with a long

frame length. The main idea is an extrapolation from Shannon's theory of communication. Shannon shows that an ultimate code would be one where a message is sent infinite times, each time shuffled randomly, but this requires infinite bandwidth so this schema is unpractical. The contribution of turbo codes is that sending the information infinite number of times is not really needed, just two or three times provides pretty good results.

3.5.1.1 Turbo Coding

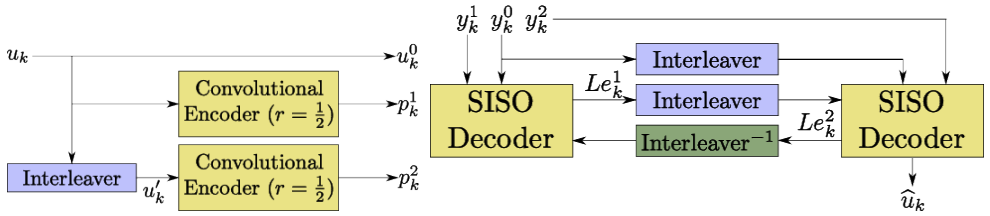
The most common turbo encoder consists of parallel concatenation of some Recursive Systematic Convolutional (RSC) encoders, each with a different interleaver, working on the same information. The purpose of the interleaver is to offer to each encoder an uncorrelated version of the information. This results in independent parity bits from each RSC. It seems logical that as a better interleaver is used, these parity bits will be more independent. The usual configuration consists of two identical convolutional encoders with rate $1/2$ and a pseudo-random interleaver, π, this schema is called a Parallel Concatenated Convolutional Code (PCCC). Figure 3.5a shows the block diagram of a turbo encoder with its two constituent convolutional encoders.

The input bits u are grouped in sequences whose length N is equal to the size of the interleaver. The sequence u' is obtained as the result of the interleaving process. The first encoder receives the sequence u and produces the pairs (u_k, p_k^1) and the second encoder receives the sequence u' and produces the pairs (u'_k, p_k^2). Since both encoders are systematic encoders $u'_k = \pi(u_k)$, and, as π is known by the decoder, only (u_k, p_k^1, p_k^2) will be transmitted. The rate of this encoder is $1/3$ but it can be increased by puncturing by $1/2$.

3.5.1.2 Turbo Decoding

Turbo decoding is based on an iterative process to improve performance and it uses, as a basic decoder unit, a Soft-Input Soft-Output algorithm. The block scheme of a common turbo decoder is shown in figure 3.5b.

First of all, the sequence encoded by the first encoder is decoded by the first decoder as in an usual convolutional code scheme. As a result, this decoder returns soft information, that is to say, an estimation about which were the values of the bit in the original sequence and how likely is this estimation for each bit. This information is called extrinsic information in the literature of turbo codes. The extrinsic information of the first decoder is interleaved in the same manner that the input bits had been interleaved in the turbo encoder before they are applied to the second encoder. The next step is to send this interleaved information to the second decoder. This decoder



(a) Common turbo encoder (b) Common turbo decoder

Figure 3.5: Dual Turbo Encoder/Decoder with ratio $r = \frac{1}{3}$.

takes the extrinsic information of the first decoder into account when it decodes the sequence encoded by the second encoder and gives a new estimation about the original values. This process is repeated several times depending on the performance that is required of the system. On the average, 7 or 8 iterations give adequate results and no more 20 are ever required.

There are some algorithms that can be modified to use as a turbo decoder component but the ones most used are the Soft Output Viterbi Algorithm [50, 52] and the BCJR [2] or Maximum A-posteriori Probability (MAP) algorithm. SOVA is a combination of iterative decoding with a modified form of Viterbi decoding and it maximizes the probability of a sequence. On the other hand, MAP maximizes the output probability based on some knowledge of the input *a priori* probabilities and soft output from the demodulator.

3.5.2 Turbo Fingerprinting Scheme

The major contributions of the turbo fingerprinting scheme, presented by Zhang *et al.* in [121] with regard to the Boneh-Shaw's scheme are the reduction of codeword length by means of the use of turbo codes as outer code and the improvement of decoding the decoding runtime by a Maximum Likelihood Decoding algorithm.

3.5.2.1 Concatenated Code

The proposed scheme consists of a concatenated turbo code with a Boneh-Shaw code, that is, each symbol that a turbo encoder generates is coded by a Boneh-Shaw encoder. Formally, Zhang *et al.* define their code $\Psi(L, N, n, d)$ as the concatenated code that results of the composition of an outer (n_0, k_0)-turbo code and an inner Boneh-Shaw $\Gamma_0(n, d)$-code, where L is a turbo code length and N is the users' number.

The first step in the process is to generate a random binary string that will be the user identification $m(u_i)$ for the user u_i, where $1 \leq i \leq N$. Next, $m(u_i)$ is divided into L groups of k_o bits each one. This groups are encoded by an (n_o, k_o)-turbo encoder and a sequence of $L \times n_0$ bits is produced. The output binary sequence is represented by

$v = v_1 v_2 \cdots v_L$ where each group v_j is constituted by n_0 bits. Each v_j is coded by the inner code $\Gamma_0(n,d)$ where, for design reasons, n must satisfy the condition $n \geq 2^{n_0}$. As a result, the sequence $W^{(v)} = W^{(v_1)} \parallel W^{(v_2)} \parallel \cdots \parallel W^{(v_L)}$ is obtained, where $W^{(v_j)}$ is the codeword of $\Gamma_0(n,d)$-code assigned to v_j.

To formalize the encoding process, Zhang *et al.* define the $\Psi(L,N,n,d)$ encoding algorithm as follows:

Algorithm 3.4. *$\Psi(L,N,n,d)$ encoding algorithm defined in [121]:*
Let $m(u_j)$ be the identification of user u_j $(1 \leq j \leq N)$

1. $v = Turbo - Encoding(m(u_j))$
2. *For each* $1 \leq k \leq L$
   $$W^{(v_k)} = \Gamma_0(n,d) - Encoding(v_k)$$
3. *Let* $W^{(v)} = W^{(v_1)} \parallel W^{(v_2)} \parallel \cdots \parallel W^{(v_L)}$

This process is repeated for all u_j in such a way that all users will have their own identification fingerprint. In the fingerprinting environments the common attack is the collusion attack, that is, some users compare their marked objects and produce, according to the *Marking Assumption* defined in Definition 2.3, a pirate object which contains a false fingerprint that lies in $\Gamma(C)$. The general schema for two traitors is shown in Figure 3.6.

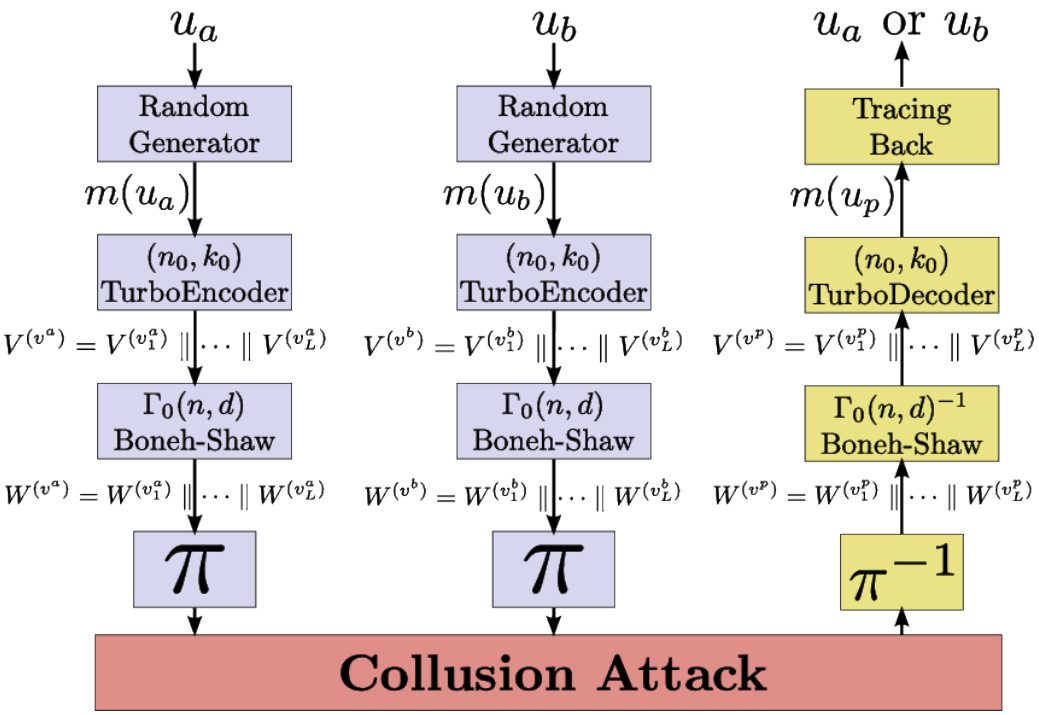


Figure 3.6: Turbo fingerprinting scheme for 2 traitors.

The aim of this kind of systems is to find, at least, one user who is part of the coalition. So the authors present Algorithm 3.5 to accomplish this purpose. The main idea of the decoding algorithm is to decode the Boneh-Shaw layer and to choose one of the symbols retrieved by this layer for this position i as the input symbol to the turbo decoder for this position i. Note that, if the Boneh-Shaw code was error-free, then for each position, the turbo decoder could choose among more than one symbol, depending on the symbols of the traitors in this position. The proposal of Zhang *et al.* was to choose one at random. A formal definition is shown by the following algorithm:

Algorithm 3.5. $\Psi(L,N,n,d)$ *decoding algorithm defined in [121]:*
Given $x \in \{0,1\}^l$*, find a subset of the coalition that produced* x*.*

1. *Apply algorithm 3.4 to each of the L components of* x*.*
   For each component $i = 1,2,\cdots,L$*, arbitrarily choose one of the outputs of algorithm 3.4.*
   Set v_j *to be this chosen output.*
   Form the word $v = v_1 v_2 \cdots v_L$
2. $m(u_j) = Turbo - Decoding(v)$
3. *Output "User* u_i *is guilty"*

3.5.3 A new critical performance analysis

To state the performance of turbo fingerprinting codes, the authors in [121] enunciate the following theorem:

Theorem 3.5. *[121] Given integers* N*,* c *and* $\epsilon > 0$*, set*

$$n = 2c$$

$$d = 2n^2(\log(2n) + m)$$

$$m = \log\left(\sum_{d_e=d_{min}}^{N} \frac{A_{d_e}}{\epsilon}\right)$$

where A_{d_e} *is the number of codewords with weight* d_e*. Then the fingerprinting scheme* $\Psi(L,N,n,d)$ *is* c*-secure with* ϵ*-error. The code contains* N *codewords and has length* $Ld(n-1)$*. Let* x *be a word which was produced by a coalition* C *of at most* c *users. Then algorithm 3.5 will output a member of* C *with probability at least* $1-\epsilon$*.*

The authors in [121] prove theorem 3.5, assuming the well known expression for the error probability P_e of turbo decoders in BSC channels (for detailed references concerning error probability of turbo decoders in BSC channels see [74, 115]). In the present scenario the channel, from the turbo codes point of view, is a Boneh-Shaw

code with error probability ϵ'. In [121], the authors express the turbo coded error probability as a function of the Boneh-Shaw code error probability. Denoting by P_e, the error probability of turbo codes in a BSC, the expression is

$$P_e \leq \sum_{d_e=d_{min}}^{N} A_{d_e} P_2(d_e) \tag{3.6}$$

where A_{d_e} is the number of codewords with weight d_e and $P_2(d_e)$ is the error probability between two codewords. Let the decoding error probability of code Γ_0 be ϵ'. The authors assume that the error probability between two codewords is smaller than the error probability of code Γ_0. So, from the authors' point of view

$$P_e \leq \sum_{d_e=d_{min}}^{N} A_{d_e} P_2(d_e) \leq \sum_{d_e=d_{min}}^{N} A_{d_e} \epsilon' \tag{3.7}$$

We now show that this is not in many cases correct.

Suppose a turbo fingerprinting code consists of an (n,k)-turbo code concatenated with a Boneh-Shaw code with negligible error probability ϵ. Moreover assume that two traitors attack this scheme by a collusion attack according to definition 2.3.

In the decoding process, the Boneh-Shaw decoder retrieves, for each position, 2 symbols with probability $\frac{2^n-1}{2^n}$ and only 1 symbol otherwise (here we suppose, as an approximation, that in a collusion of 2 users a position can not be detected with probability $\frac{1}{2^n}$. So $\frac{1}{2^n}$ is the probability that the symbol in a particular position will be the same for 2 codewords). The scheme proposed by Zhang *et al.* takes one of them at random and sends it to the turbo decoder.

As n increases, $\frac{2^n-1}{2^n}$ tends to 1, that is, the probability that the traitors, say tc_1 and tc_2, have the same symbol in a particular position tends to 0. So the Hamming distances between tc_1 or tc_2 and the pirated word, say tc_p, delivered to the turbo decoder satisfy the equation $d_H(tc_1, tc_p) \simeq d_H(tc_2, tc_p)$. If L is the length of the codeword and n is large enough, the turbo decoder takes as input a word in which half of the symbols are erroneous respect both of the two traitor codewords. And, as a result, the turbo decoder retrieves a codeword of the turbo code codebook, but this codeword is not assigned to any user. Note that in this case, the decoded codeword will be different from the pirate words with a very high probability, which is not desirable at all. In this case there cannot be a false positives because, the turbo encoded words are a random sequence (like hash functions) and the collision probability for these functions is very small.

As an example suppose a $(3,1)$-turbo code that consists of two component convolutional codes. The connection expressed in octal is $(3,1)$. The traitors have the sequences

$$t_1 = 1100010010,$$

$$t_2 = 0000111000.$$

The sequences t_1 and t_2 are turbo coded to generate

$$tc_1 = Turbo-Encoding(t_1) = 100110000001001101011010110001000000,$$

$$tc_2 = Turbo-Encoding(t_2) = 000000000001100111101011011011111100.$$

These turbo coded sequences may be expressed in octal notation as:

$$tc_1 = Turbo-Encoding(t_1) = 460115326100$$

$$tc_2 = Turbo-Encoding(t_2) = 000147533374$$

The Feasible Set will be

$$\Gamma(tc_1, tc_2) = \left(\begin{Bmatrix}4\\0\end{Bmatrix}, \begin{Bmatrix}6\\0\end{Bmatrix}, \begin{Bmatrix}0\\\end{Bmatrix}, \begin{Bmatrix}1\\\end{Bmatrix}, \begin{Bmatrix}1\\4\end{Bmatrix}, \begin{Bmatrix}5\\7\end{Bmatrix}, \begin{Bmatrix}3\\5\end{Bmatrix}, \begin{Bmatrix}2\\3\end{Bmatrix}, \begin{Bmatrix}6\\3\end{Bmatrix}, \begin{Bmatrix}1\\3\end{Bmatrix}, \begin{Bmatrix}0\\7\end{Bmatrix}, \begin{Bmatrix}0\\4\end{Bmatrix}\right).$$

After decoding the Boneh-Shaw code, if no errors are produced, a possible sequence sent to the turbo decoder is

$$tc_p = 000110000001001101101011011001111000,$$

or in octal,

$$tc_p = 060115533170.$$

After turbo decoding, the word obtained is

$$t_p = 1100001000,$$

which is none of the traitors' codewords, $d_H(t_1, t_p) = 3$ and $d_H(t_2, t_p) = 4$. That is, this construction cannot be a correct fingerprinting scheme because the system cannot trace back t_1 or t_2 from tc_p.

3.5.4 Proposed improvements and open problems

One of the most important improvements that the turbo codes have contributed to error correcting codes is the use of the likelihood of every information bit during the decoding process. The proposed improvements in this section are based on the use of this information in two different ways. The first one, uses the information provided by the fingerprinting layer to calculate the likelihood for each information bit at the first turbo decoding iteration. On the other hand, the second proposal is centered in the fact that the cross-correlation between the likelihood of the decoded bits and all possible words (users) reaches the maximum value when the evaluated user has taken part in the collusion attack, i.e. is guilty.

3.5.4.1 Decoding by the use of likelihood information in undetected coefficients

There exist some techniques, as concatenating a turbo codes with a Boneh-Shaw code or the ones proposed in [89], that can be used to detect, at the turbo decoder input, if a particular bit has been modified by a collusion attack. As it is also well-know, each constituent decoder in a turbo decoder uses the likelihood information of every information bit externalized by the other but this likelihood is not known at the first iteration by the first constituent decoder. The usual solution is to consider that all values are equally likely, that is to say, the value of $L_e^{(2)}$ for all bits in the first iteration is initialized to 0 (take into account that L_e is the Log-likelihood ratio). The first improvement is to modify the value of L_e taking into account the information of the Boneh-Shaw layer. The main idea is, as the undetected bits by the traitors during the collusion attack are known, the decoder can assign a greater likelihood to these bits in the first decoding stage. After few simulations, it can be concluded that a little improvement around 2% appears if the initial value of L_e is slightly modified by the use of this previous information. Note that, when an error is produced during the decoding process, the returned word identifies one legal user which has not taken part in the collusion, that is a false positive. In other words, in this situation the decoding process frames an innocent user. In a correct TFC system, a bit error probability around 0 is needed. If the value of L_e value is highly altered, the effect can be counterproductive because, in this case, the turbo decoder does not converge correctly. This is shown in figure 3.7.

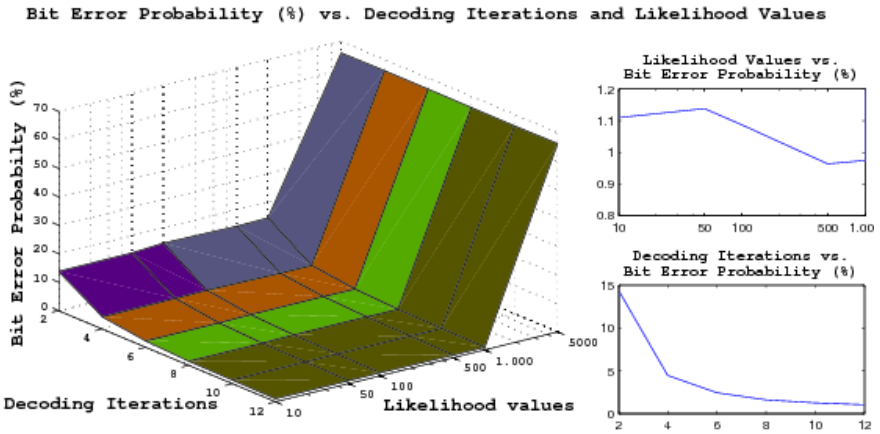


Figure 3.7: Bit probability error of a TFC with generator sequences constituent RSC $(53,75)_8$ over collusion attack decoded using likelihood information.

Even though this improvement has been applied to TFC, it is not sufficient to guarantee that the probability of finding one of the traitors will be small enough. It seems that the error probability has been a slight improvement and it can be reduced near to 0 by the use of some block error correcting code as BCH. The figures 3.8a, 3.8b and 3.8c show the results of the use of a (15,11) Hamming code, a BCH(127,64) and

a BCH(127,22) respectively, concatenated with a turbo code decoded taking into account the likelihood information.

Some open problems are how to modify the channel characteristic, in turbo code notation that is the value of L_c, taking into account the positions not detected by the attackers and the study of the relation between the RSC generator sequences and the likelihood value to assign to each bit at first decoding stage.

3.5.4.2 Decoding by the use of making correlation conditional on likelihood

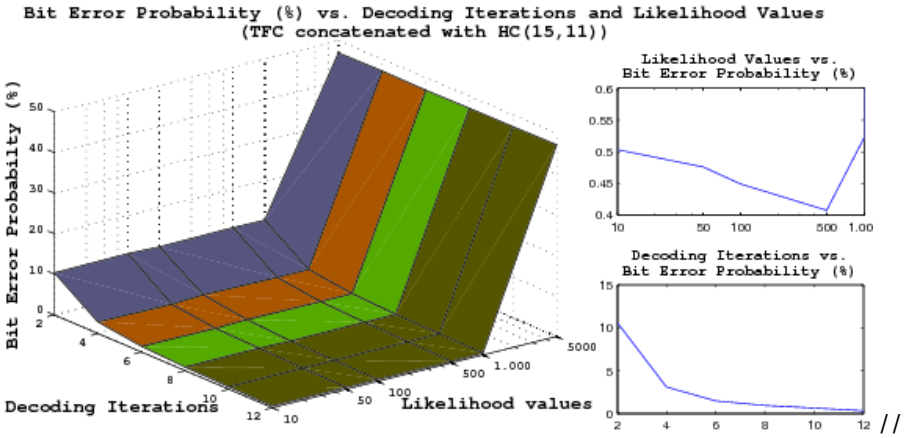
The decoding algorithm proposed in the original paper of TFC was the commonly used to decode turbo codes. The main problem was that the turbo decoder returns the most likely codeword over all the code space; that is, if a user ID of 512 bits is used, the turbo decoder will return the word of 512 bits that is the most likely to have been sent. This means that with a very high probability an innocent user has been framed. This is the main reason of the problems produced in the decoding stage of TFC.

In practice, it is very difficult to think about an application in which 2^{512} users are needed. For instance, Figure 3.9 shows the results of a 1000 iterations simulation of one system which uses a TFC with user ID of 128 bits but the system has only 1000 users whose user IDs are randomly distributed in the codes pace. In each iteration two different users are chosen randomly and a collusion attack is performed with their codewords. Next, this colluded codeword is decoded by the turbo decoder in order to obtain one of the traitors. None of them have been found by the use of the original TFC decoding system or, as is named in the figure, TFC without correlation. If the word which results from the TFC original decoding system is correlated with all possible user IDs, we will always find at least one of the traitors and, more than 90 percent of the times, the two traitors will be found. If the likelihood information is used instead of the pirate word, the probability of finding the two traitors comes close to 100 percent. In other words, the user IDs that have the maximum correlation value with the likelihood returned by the turbo decoder are the user IDs of the members of the collusion.

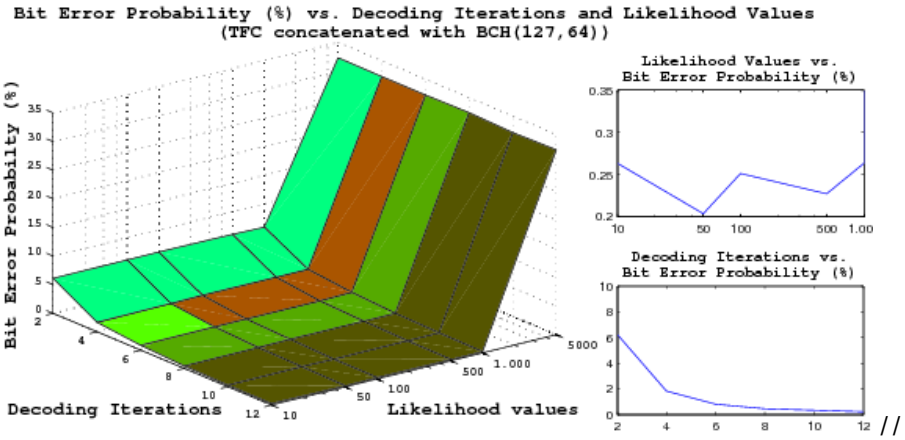
The main drawback of this proposal is that the decoding time increases exponentially with the number of users.

3.6 Conclusions

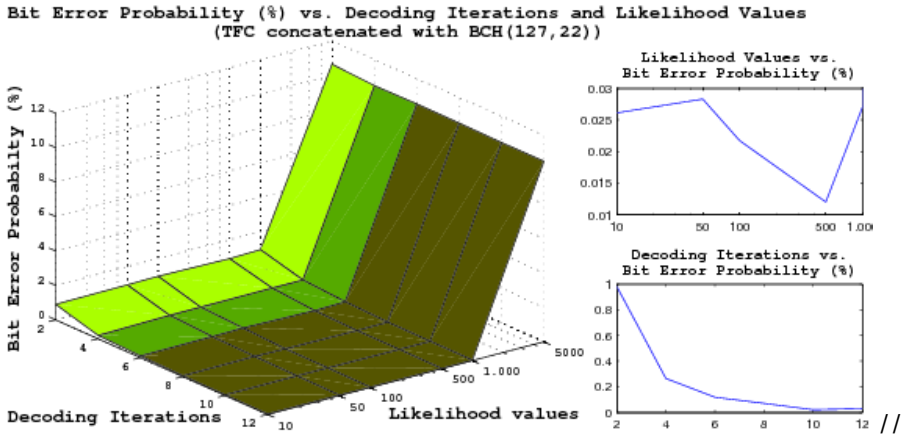
On one hand, this chapter shows a new problem in Collusion Secure Convolutional Fingerprinting Information Codes: False positives. As a result of the analysis of the work by Zhu *e*t al. in [122], the drawbacks of not considering false positive have been enlightened. The original results in [122] are revisited from the point of view of the false positive problem. Moreover the probability of false positives has been quantified



(a) TFC concatenated with HC(15,11).

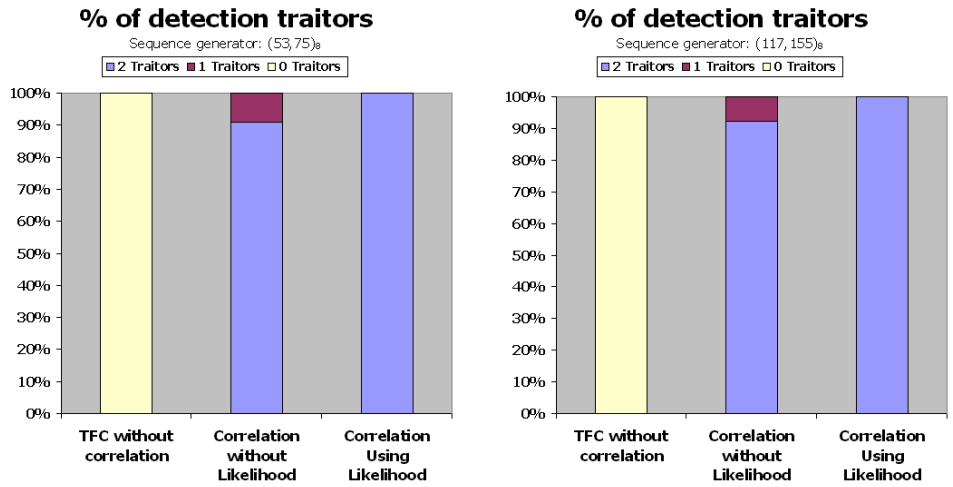


(b) TFC concatenated with BCH(127,64).



(c) TFC concatenated with BCH(127,22).

Figure 3.8: Bit probability error of a TFC with generator sequences constituent RSC $(53,75)_8$ concatenated with several error correcting codes over collusion attack decoded using likelihood information.



(a) TFC with generator sequences constituent RSC $(53,75)_8$

(b) TFC with generator sequences constituent RSC $(117,155)_8$

Figure 3.9: % of detecting 0, 1 or 2 traitors after a collusion attack of 2 traitors by the use of TFC with correlation decoding.

formally and contrasted with simulations. Finally some guidelines for a correct design of Collusion Secure Convolutional Fingerprinting Information Codes are given.

On the other hand, this chapter discusses an undesired problem in the analysis of the turbo fingerprinting codes presented by Zhang *et* al. in [121]. We show that the probability of tracing one of the traitors tends to 0 when the alphabet size of the outer turbo code increases. That is because the symbol-by-symbol collusion attack performed by pirates is not treated efficiently by the decoding algorithm proposed in [121]. Note that, from the point of view of the turbo decoder, the error probability of the equivalent channel tends to 1/2, because it takes as input symbols one of the symbols retrieved by the Boneh-Shaw decoder chosen at random.

The new problem found in the turbo fingerprinting codes renders them inapplicable in many cases unless the design takes into account our new contribution. Moreover, our studies indicate that, the more efficient the turbo fingerprinting code design is, from the point of view of the length requirement, a far worse performance is obtained from the tracing algorithm. In other words, to find a traitor will be more complicated when the (n,k)-turbo code used, has large values of n.

Besides, two different ways to improve the performance of turbo fingerprinting codes are given. These two ways use the likelihood of the turbo decoder to perform the improvements. The first proposal modifies this likelihood at the input of the turbo decoder and the other use the turbo decoder output likelihood to correlate it with the user IDs in order to find the traitors. Moreover, this two improvements can be

integrated in the same scheme.

CHAPTER 4

USE OF TURBO CODES WITH LOW-RATE CONVOLUTIONAL CONSTITUENT CODES IN FINGERPRINTING SCENARIOS

Since the introduction of turbo codes in 1993, many new applications for this type of codes have arised. In this chapter, a family of turbo codes that can be used as fingerprinting codes is presented. These fingerprinting codes are secure against attacking coalitions of size 2. The family discussed has as its constituent codes a type of low-rate convolutional codes with maximum free distance. These low rate convolutional codes are commonly used in code-spread Code-Division Multiple-Access (CDMA) applications. Moreover, how efficient traitor tracing can be performed by using the turbo decoding algorithm is shown. Furthermore, the performance of the developed scheme has been studied in the presence of a watermarking layer which adds noise to the codewords.

4.1 Introduction

The construction of collusion secure codes was first addressed in [11]. In that paper, Boneh and Shaw obtain ($c > 1$)-secure codes, which are capable of identifying a guilty user in a coalition of at most c users with a probability ϵ of failing to do so. The construction composes an inner binary code with an outer random code. Therefore, the identification algorithm involves the decoding of a random code, that is known to be a NP-hard problem [6]. Moreover, the length of the code is considerably large for small error probabilities and a large number of users.

Barg, Blakley and Kabatiansky in [6] used algebraic-geometric codes to construct fingerprinting codes of positive rate to reduce the decoding complexity. Moreover, their proposal reduces the decoding complexity to $O(poly(n))$ for a code length n.

The case of 2 traitors is of particular interest and has been extensively discussed in the literature [38, 91, 31].

As it has been discussed in the previous chapter, one of the first approaches to construct fingerprinting codes using convolutional codes was the Collusion Secure Convolutional Fingerprinting Information Codes presented in [122]. These codes have shorter information encoding length and achieve optimal traitor searching in scenarios with a large number of buyers. Unfortunately, these codes suffer from an important drawback in the form of false positives, in other words, an innocent user can be tagged as guilty with very high probability. In [TBFS09], these codes are analyzed in depth and the probability of false positives is quantified. As an evolution of convolutional fingerprinting codes, turbo fingerprinting codes were proposed in [121]. In [TBFS08a], an analysis of practical implementation problems of these codes was presented. This analysis shows that turbo fingerprinting codes must obey to some restrictions in order to obtain the desired performance and that the problem of false positives remains. Moreover, in order to obtain a good performance of high-rate turbo codes, it was necessary to use a matching filter so, the decoding time is highly time consuming.

The first really implementable proposal of turbo-like codes in fingerprinting scenarios was presented by Jourdas and Moulin in [66]. In this proposal, a turbo-encoder composed by low-rate convolutional codes is utilized. Moreover, they separate the turbo-encoder output in blocks that are transformed to the Discrete Cosine domain in order to obtain a spherical code. On the decoding side, these codes do not use a typical turbo-decoder because extrinsic information is not interchanged by the convolutional decoders. They use list-Viterbi decoders in order to obtain a list of possible traitors for each convolutional code. All these lists are concatenated, re-encoded and a matching filter is applied between this list and the channel output. The codeword of the list which has the maximum correlation with the channel output is selected as a traitor.

4.1.1 The novel contribution

In this chapter, a family of fingerprinting codes based on turbo codes is constructed. Intuitively the use of turbo codes is justified taking into account the well known performance of turbo codes in noisy scenarios which is the case of watermarking processes when fingerprints are embedded into digital content.

A class of Maximum Free Distance Low-Rate Convolutional Codes (MFDLRC) [42] were designed by Frenger *et al.* in order to be used as code-spread in Code-Division

Multiple-Access (CDMA) scenarios. The goal of this technique is to avoid the interference produced by the users of a mobile communication system to the signal of a particular user. It is easy to see that a collusion attack in fingerprinting scenarios produces a similar effect to the codeword of one traitor when the pirate word is generated. With this in mind, it seems a good option the use of MFDLRC as part of a fingerprinting scheme. Therefore, MFDLRC will be used as the constituent codes of turbo codes that lie at the core of the presented family of fingerprinting codes.

In practical scenarios, the performance of the presented fingerprinting scheme could be degraded by the presence of a watermarking layer, that is to say, the embedding process will clearly affect the achievement of the fingerprinting scheme goal. As stated above, a second goal was to simulate and quantify the degradation produced by the presence of a watermarking layer.

The chapter is organized as follows. In section 4.2, some definitions on fingerprinting, error correcting codes and turbo codes are reviewed and maximum free distance low-rate convolutional codes are presented. Section 4.3 discusses the code construction of turbo fingerprinting codes and defines a family of this kind of codes. The results of the proposed system are presented in section 4.4. Moreover, in the same section,the watermarking layer has been included and its impact has been measured. Finally, some conclusions are given in section 4.5.

4.2 Definitions and previous results

Let a *code* be a set of n-tuples of elements from a set of scalars called the *code alphabet*. The elements of the code are called *codewords*. If the code alphabet is a finite field $\mathbb{F}_q$, then a code C is a *linear code* if it forms a vector subspace. The dimension of the code is defined as the dimension of the vector subspace. Let $\mathbf{a},\mathbf{b} \in \mathbf{F}_q^n$ be two words, then the *Hamming distance* $d(\mathbf{a},\mathbf{b})$ between $\mathbf{a}$ and $\mathbf{b}$ is the number of positions where $\mathbf{a}$ and $\mathbf{b}$ differ. Let C be a code, the *minimum distance* of C, d, is defined as the smallest (Hamming) distance between two different codewords. A code C of length n with M codewords will be denoted as $C(n,M)$. A linear code with length n, dimension k and minimum distance d over the field $\mathbb{F}_q$ is denoted as an $[n,k,d]_q$-code, or simply as an $[n,k,d]$-code.

A convolutional code (see figure 4.1) is a type of error-correcting code in which each k-bit information symbol (each k-bit string) to be encoded is transformed into an n-bit symbol, where k/n is the code rate ($n \geq k$) and the transformation is a function of the last K information symbols, where K is the constraint length of the code. Every one of this functions could be denoted by a generator polynomial which represents the relation between these K information symbols and a single output bit. The minimum

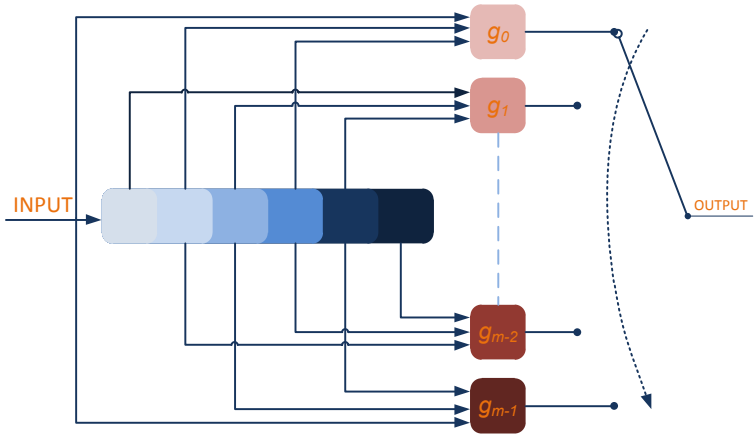


Figure 4.1: Feed-forward concolutional encoder for a rate $R = 1/m$ and constraint length $k = 7$.

free distance d_{free} of a convolutional code is defined as

$$d_{free} \triangleq \min\{d(v', v'') : u' \neq u''\} \tag{4.1}$$

where v' and v'' are the codewords corresponding to the information sequences u' and u''.

4.2.1 Turbo Codes

Turbo codes were introduced in 1993 by Berrou, Glavieux and Thitimajashima [9], [10]. Turbo decoding is based on an iterative process. There are several algorithms that can be modified in order to be used as the underlying routines in a turbo decoder. The ones that are mostly used are the Soft Output Viterbi Algorithm (SOVA) [52] and the BCJR [2] or Maximum A-posteriori Probability Algorithm (MAP). The MAP algorithm maximizes the likelihood of each output bit based on its knowledge of the input *a priori* probabilities and the soft output from the demodulator. The Max-Log-MAP optimization [51] can be used instead of MAP.

4.2.2 Maximum free distance low-rate convolutional codes

The codes presented in [42] are obtained from feed forward convolutional encoders which give minimum information error weight. In order to compare the performance of feed-forward convolutional encoders, the *Superior Distance Spectrum* is used. The sum of the bit errors for error events at distance d will be called the information error weight, and denote it by c_d. Given a code with free distance $\widetilde{d}_f$, a feed forward encoder with information error weights c_d has a *distance spectrum superior* to a feed forward encoder with error weights $\widetilde{c}_d$, if one of the following conditions is fulfilled:

1. $d_f > \widetilde{d}_f$ or,

2. $d_f = \widetilde{d}_f$ and there exists an integer $l \geq 0$, such that $c_d = \widetilde{c}_d$, for $d = d_f, d_f + 1, \ldots, d_f + l - 1$ and $c_d < \widetilde{c}_d$, for $d = d_f + l$.

An Optimum Distance Spectrum (ODS) convolutional code is a code generated by a feed-forward encoder that gives a Superior Distance Spectrum (SDS) when is compared to the rest of codes with the same rate and constraint length.

Unfortunately, there is not an efficient mechanism to find these codes so in [42] the authors use some exhaustive computer-search techniques to obtain convolutional encoders that give ODS for code rates of 1/2, 1/3 and 1/4. Then, they apply a nested encoder search, which consists of adding new polynomials to an 1/4 ODS encoder in order to obtain a convolutional encoder of ratio $R = 1/5$. This new encoder is a $R = 1/4$ ODS descendant encoder. By means of repeating this search process, any $R = 1/n$ encoder could be generated. In this way, a set of polynomials $g = \{g_0, g_1, \ldots, g_{n-1}\}$ will be obtained. At the point when the process becomes excessively time consuming, no new polynomials are added. Instead, the previously found generators are applied again.

4.2.3 Traceability Codes

When the fingerprint is embedded by means of a watermarking layer, it is represented with a little modification of a real value. If the attackers detect the watermarked position (by a collusion attack), they can change this value to a random value.

The collusion attack described in Section 4.1 is modeled by the construction of a descendant, if the descendant is taken as being the word in the pirate copy and the parents as being the codewords of the colluders.

It is clear that in the context of marking digital content the codes of interest are those defined over $\mathbb{F}_2$. Unfortunately, for $\mathbb{F}_2$, we have to allow for some error probability in the outcome of the tracing process. In order to achieve an error probability as small as desired a single code is not sufficient and a *family of codes* is needed [11, 6, 107]. Below, such a family of codes will be denoted as $\mathscr{F} = \{F_t : t \in \mathscr{T}\}$, where $\mathscr{T}$ is a finite set. Furthermore, randomness will be also needed. The family $\mathscr{F}$ will be publicly known, but the specific code F_t that it will be used, will be chosen at random from the codes in $\mathscr{T}$ with probability $p(t)$. This choice of F_t must be kept secret. Below it will assume that $p(t) = |\mathscr{T}|^{-1}$ for all $t \in \mathscr{T}$. The elements in the set $\mathscr{S}$ are usually called *keys*. For a given family of codes to achieve exponential decline of the error probability, the number of keys must grow exponentially with the length of the code [6].

Moreover, a *tracing algorithm* for the family of codes $\mathscr{T}$ must be defined. For a family of binary fingerprinting codes $\mathscr{T}$, a tracing algorithm A is a function from the

set of descendants and the set of keys to the subsets of codewords (coalitions) of the codes in $\mathscr{T}$. In other words, given a descendant a tracing algorithm will allow us to identify at least one of the parents of the descendant, with high probability.

Definition 4.1 (Family of binary fingerprinting codes [11])**.** *Let $\mathscr{T}$ be a finite set of elements called keys. A family of binary fingerprinting codes $\mathscr{F} = \{F_t : t \in \mathscr{T}\}$, is c-secure with ϵ-error, for $c \geq 2$, if there exists a tracing algorithm A that satisfies the following condition: if a coalition U of size at most c creates a descendant $\mathbf{z}$, then*

$$\Pr(A(\mathbf{z}, s) \subseteq U) > 1 - \epsilon,$$

where the probability is taken over the random choices made by the coalition when creating the descendant and over the keys $t \in \mathscr{T}$.

4.3 Family of turbo fingerprinting codes for coalitions of size two

We start our contribution in this section. We will first argue our choice for the underlying turbo code construction, and then we show how to fit this construction into the definition of a family of fingerprinting codes.

4.3.1 Code construction

From the discussion in Section 4.2.1, a turbo code consists of l convolutional encoders and $l-1$ interleavers. In order to present our code construction, we must therefore specify a way to select these constituent convolutional encoders.

Given an ODS feed-forward convolutional encoder of rate $R = 1/n$ and constraint length K constructed with the set of generator polynomials $\mathscr{G} = \{g_0, g_1, \ldots, g_{n-1}\}$ where $g_i \neq g_j$ if $i \neq j$. Let $I = (i_0, \ldots, i_{u-1})$ where $i_j \in \{0, \ldots, n-1\}$. An *ODS descendant encoder* is defined as a feed-forward convolutional encoder of rate $R = 1/u$ and constraint length K with generator polynomials $\{g_{i_0}, g_{i_1}, \ldots, g_{i_{u-1}}\}$ where each g_{i_j} is a member of the set $\mathscr{G}$. In other words, the convolutional encoders of our turbo fingerprinting codes will be generated as the descendant encoders of ODS convolutional encoders. Note that for a given ODS convolutional encoder of rate $R = 1/n$ and constraint length K, we can generate u^n different ODS descendant encoders of rate $R = 1/u$ and constraint length K. Even though it is possible to use other convolutional code constructions and trellis termination methods, we use recursive systematic encoders and uninterleaved dual termination because it is easier to implement. Fortunately enough, for any feed-forward code with generator polynomials $g = \{g_0, g_1, \ldots, g_{n-1}\}$, there exists an equivalent recursive (feed-back) systematic code with generator polynomials $g_f = \{1, \frac{g_1}{g_0}, \ldots, \frac{g_{n-1}}{g_0}\}$ [74].

Construction 4.1. *Let C_O^n be an ODS convolutional encoder of rate $R = 1/n$ and constraint length K. Let $\mathscr{D}_O^u$ be the set of all ODS descendant encoders of rate $R = 1/u$ of C_O^n. The set $\mathscr{S}_O^u$ is the set of systematic encoders, each being equivalent to an element in $\mathscr{D}_O^u$. Observe that $n^{u-1} \leq |\mathscr{S}_O^u| \leq |\mathscr{D}_O^u| = n^u$. To construct an encoder of rate $R_{tc} = 1/(lu - l + 1)$, we choose l encoders from the set $\mathscr{S}_O^u$. Note that the encoders need not be different. Moreover, in order to improve the ratio, a puncture of systematic bits is done. This is the reason for the $l-1$ term in the R_{tc} equation. It is clear that with this procedure we can construct, at least, $n^{(u-1)l}$ turbo fingerprinting codes. We denote this set by $\mathscr{T}_l$.*

4.3.2 Family construction

We now show how to obtain a family of turbo fingerprinting codes. This will be done using the set $\mathscr{T}_l$ presented above and Definition 4.1. To this end, it will be useful to index the elements in $\mathscr{T}_l$. By using an arbitrary total order relation we denote each element of $\mathscr{T}_l$ by an integer in $\mathscr{K} = \{0, \ldots, |\mathscr{T}_l| - 1\}$ where $|\mathscr{T}_l|$ denotes the number of elements of $\mathscr{T}_l$. Note that $n^{(u-1)l} \leq |\mathscr{T}_l| \leq n^{ul}$. The entire set of generator polynomials will be publicly known. The subset of generator polynomials and the number of constituent encoders will be a part of the key and will be kept secret.

As a tracing algorithm we will use the Max-Log-Map algorithm. As it will be show below, given a descendant (pirate) word, the iterative decoding of the constituent encoders by means of Max-Log-Map algorithm will output one of the parent (traitor) codewords with high probability. More precisely, the family of turbo fingerprinting codes is defined as:

Definition 4.2. *The set $\mathscr{T}_l$, of turbo encoders in Construction 4.1, using as a tracing algorithm the iterative decoding Max-Log-Map algorithm is a family of 2-secure turbo fingerprinting codes with ϵ-error. The elements in the set $\mathscr{K}$ are the secret keys.*

In order to use our family we draw an element of $\mathscr{K}$, say s, with uniform probability. This choice must be kept secret. The fingerprints assigned to the authorized users will be the codewords of the code $\mathscr{T}_l^s$. Given a descendant from a coalition of at most 2 users, the codeword at the output of the Max-Log-Map algorithm will be identified as a traitor.

4.4 Security analysis

The analysis is restricted to F_2 for simplicity and to allow comparison with other well-known F_2 techniques. This scenario is shown in Figure 4.2. There are two users with identifiers $UserID_i$ and $UserID_j$. We assign codeword $\mathbf{u}^i = \{u_1^i, \cdots, u_n^i\}$ to user

$UserID_i$ and codeword $\mathbf{u}^j = \{u_1^j, \cdots, u_n^j\}$ to user $UserID_j$ respectively. The k-th position of the codeword for user j is denoted by u_k^j. The codewords $\mathbf{u}^i$ and $\mathbf{u}^j$ belong to a turbo fingerprinting code. This turbo fingerprinting code has a fixed selection of generator polynomials (the secret *keys*).

When these two users perform a collusion attack they generate a pirate (descendant) word $\mathbf{z}$. In this paper, as in [66, 57], we consider the following attack: when $u_k^i = u_k^j$ the position remains undetectable so $z_k = u_k^i = u_k^j$. When $u_k^i \neq u_k^j$ the position becomes detectable and the traitors decide to erase the position, that is to say, $z_k = *$ where, as discussed in Section 4.2.3, "$*$" denotes an erasure. Moreover, the traitors are allowed to add additive white gaussian noise to the pirate, in order to increase "confusion". Therefore the result of the collusion attack is a pirate word $\widehat{z} = \frac{1}{c}\sum_{k=1}^{c} u^k + n$ where c is the number of colluders, u^k is the fingerprint sequence for the user k and n is the noise component. At this point an observation is probably in order. This kind of attack has been considered in conjunction with the existent literature [66, 57]. Moreover, the reason for not considering other kinds of attacks in the watermarking layer is because the aim of this work is to study the behaviour of fingerprinting codes. All classical fingerprinting schemes [6, 11, 107] rely on the *marking assumption*. This means that the attackers can only modify the positions they detect by comparison of their copies. Therefore, for the marking assumption is mandatory to make use of an ideal watermarking layer. This assumption, of course, protects the fingerprinting code from any image processing attacks. Only recently, and in order to bring watermarking into the picture, some authors [66, 57] assume what is known as *distortion attack*. In this case, by allowing the attackers to introduce Gaussian noise, one can deal with a more realistic scenario. This assumption will also be made in this paper.

In a practical scenario the distributor would retrieve $\widehat{z} = z + n$, and apply a tracing algorithm. In this case the identification routine consists in the Max-Log-MAP iterative decoding algorithm. The simulations below show that with the considered attack, the output of the Max-Log-MAP algorithm identifies one of the traitors ($UserID_i$ or $UserID_j$) with high probability. Therefore, the algorithm will fail when the output does not match with any of the traitors, so it must be considered an error. This simulation error probability is represented by P_e. Note that, the presented results have been obtained from Monte Carlo simulations with $N_{test} = 100000$, so the estimation error must be added to P_e in order to obtain the confidence interval of the total error probability. This estimation error could be approximated by $\pm z_{1-\alpha/2}\sqrt{(1-P_e)P_e/N_{test}}$ assuming a normal distribution approximation, where $z_{1-\alpha/2}$ is the standardised normal value. In order to obtain the 95% confidence interval, we will use $z_{1-\alpha/2} = 1.96$ to calculate the range of the confidence intervals showed in the results of this paper.

Besides the theoretical results (subsections 4.4.1, 4.4.2 and 4.4.3), the proposed codes have been tested in an almost totally realistic scenario (described in 4.4.4). In

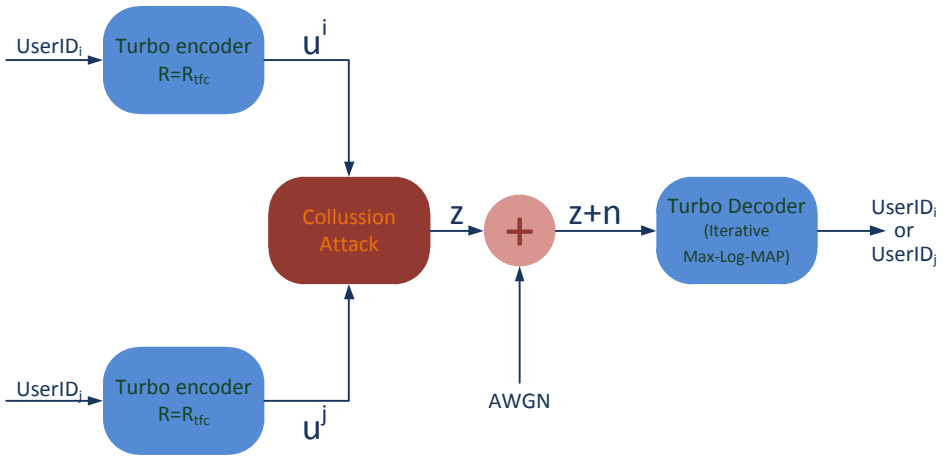


Figure 4.2: Schema of the simulated scenario.

this scenario, 2 layers can be identified. On one hand, the fingerprinting layer which is composed by the turbo fingerprinting encoder/decoder presented in the section below. On the other hand, a watermarking layer has been developed in order to embed the fingerprints into MPEG2 videos. In the same way, the extraction has been performed and evaluated (as it is explained in 4.4.5 and 4.4.6).

4.4.1 Study about the performance of the presented codes depending on constituent codes and the number of supported users

On one hand, in order to simulate different and representative codes, 5 different codes have been selected. All of them consist of two identical convolutional codes ($l = 2$). Moreover, we work with different ratios for these convolutional codes. Depending on the different internal convolutional encoder ratio, the generator polynomials selected for each family are:

- $R = \frac{1}{25}$ with generator polynomials $F_{s_1} : s_1 = \{3 \times 117, 2 \times 123, 127, 133, 6 \times 135, 2 \times 137, 145, 3 \times 155, 157, 2 \times 171, 2 \times 173, 175\}_o$
- $R = \frac{1}{20}$ with generator polynomials $F_{s_2} : s_2 = \{2 \times 117, 123, 127, 133, 5 \times 135, 137, 145, 2 \times 155, 157, 2 \times 171, 2 \times 173, 175\}_o$
- $R = \frac{1}{15}$ with generator polynomials $F_{s_3} : s_3 = \{2 \times 117, 123, 127, 4 \times 135, 137, 145, 2 \times 155, 157, 171, 173\}_o$
- $R = \frac{1}{10}$ with generator polynomials $F_{s_4} : s_4 = \{117, 127, 2 \times 135, 137, 145, 2 \times 155, 171, 173\}_o$

- $R = \frac{1}{5}$ with generator polynomials F_{s_5} : $s_5 = \{117,\ 127, 2 \times 135, 137, 145, 2 \times 155, 171, 173\}_o$

Note that $a \times b$ means the polynomial b (octal form) is used a consecutive times. For instance, the term 2×155 signifies that the polynomial 155_o (which corresponds to binary 1101101_2 and whose generator polynomial is $g(D) = 1 + D + D^3 + D^4 + D^6$) is used 2 consecutive times.

On the other hand, we also deal with the number of users that the code can support. Different constructions with different number of users have been taken into account. It is important to stress that, in the presented family of codes, the length of the code increases as $O(log_2(n))$.

Simulation results obtained considering a collusion of 2 attackers ($c = 2$) Table 4.1 shows the results of the performed simulations using a turbo fingerprinting code with $SNR = 10dB$ and constraint length $K = 7$. Note that, in these simulations, the number of decoding iterations has been fixed to 10. In this table, k is the number of bits used to represent the $UserID$. That is to say, for k input bits, 2^k users will be supported by the system. The number of traitors is $c = 2$. The value R represents the ratio of the internal convolutional encoders, therefore a ratio of $R = \frac{1}{25}$ means that the entire turbo fingerprinting code has an effective ratio of $R_{tfc} = \frac{1}{49}$. In addition to the innocent-user framing probability (P_e), the number of iterations (N_{test}) made is also specified. It is a well known fact that turbo codes are good error-correcting codes against gaussian noise. Due to the nature of the collusion attack, the errors introduced in the pirate word are different from errors produced by gaussian noise. These simulations show that the error probability of identifying a traitor increases for large values of k. This is due to the fact that as k increases the number of positions detected by the colluders also increases. Therefore, a collusion attack produces a number of errors that is larger than the number of errors that gaussian noise would produce in a communications channel.

Simulation results obtained considering a collusion of 3 attackers ($c = 3$) Figure 4.3 shows how the P_e decreases when the SNR of the channel increases, that is, when the amount of noise added by the attackers decreases. As expected, the lower the ratio is (more redundancy), the better performance is obtained. As it can be seen, it appears an unavoidable error floor. The main contribution to this error floor seems to come from the large amount of errors that a coalition of size 3 (as opposed to size 2) introduces to the pirate word. This error floor could be reduced using turbocodes constructed with convolutional codes with ratios lower than $\frac{1}{25}$. However, simulations with these values ratios cannot been performed due to a limitation of the software used to perform these simulations.

Table 4.1: Simulation results obtained using a turbo fingerprinting code with $c = 2$, $SNR = 10dB$, constraint length $K = 7$. The turbo fingerprinting code has two identical convolutional codes with generator polynomials F_{s_1}, F_{s_2}, F_{s_3}, F_{s_4}, F_{s_5}.

Ratio	k	N_{test}	P_e	***estimation error***
$R = \frac{1}{25}$	64	100000	0.00029	$\pm$ 0.000106
	32	100000	0.00002	$\pm$ 0.000028
	16	100000	0.00003	$\pm$ 0.000034
	8	100000	0.00001	$\pm$ 0.000020
$R = \frac{1}{20}$	64	100000	0.00025	$\pm$ 0.000098
	32	100000	0.00009	$\pm$ 0.000059
	16	100000	0.00005	$\pm$ 0.000044
	8	100000	0.00001	$\pm$ 0.000020
$R = \frac{1}{15}$	64	100000	0.00040	$\pm$ 0.000124
	32	100000	0.00006	$\pm$ 0.000048
	16	100000	0.00007	$\pm$ 0.000052
	8	100000	0.00001	$\pm$ 0.000020
$R = \frac{1}{10}$	64	100000	0.00042	$\pm$ 0.000127
	32	100000	0.00014	$\pm$ 0.000073
	16	100000	0.00010	$\pm$ 0.000062
	8	100000	0.00003	$\pm$ 0.000034
$R = \frac{1}{5}$	64	100000	0.00117	$\pm$ 0.000212
	32	100000	0.00043	$\pm$ 0.000128
	16	100000	0.00035	$\pm$ 0.000116
	8	100000	0.00002	$\pm$ 0.000028

4.4.2 Length comparison with other well-known fingerprinting constructions

Our aim in this paper is to improve on existing constructions in order to provide a practical fingerprinting scheme that allows tracing the guilty users efficiently. In this section we compare the length of our codes with the most studied codes in the literature: Boneh-Shaw [11] and Tardos [107]. For Boneh-Shaw codes, let N be the number of possible authorized users. Then given $\epsilon > 0$, $L = 2c\log(2N/\epsilon)$ and $d = 8c^2\log(8cL/\epsilon)$ the length of the code is $l = 2Ldc = 32c^4\log(2N/\epsilon)\log(8cL/\epsilon)$. For Tardos codes the expression of the code length is given by $l = 100c^2\ln N/\epsilon$. The comparison results considering a collusion of 2 and 3 attackers are shown in Table 4.2 and Table 4.3, respectively. Note that Boneh-Shaw codes and Tardos codes are asymptotically good codes. Therefore, they are able to achieve a very small error probability even with a large number of colluders. Nevertheless, in a real life scenario with a small number of users is reasonable to deal with 2 or 3 colluders. For instance, suppose that the digital objects to be protected are evidences in a trail or sensitive documents from a company administration board, or even preview copies of a film delivered to the critic. In these

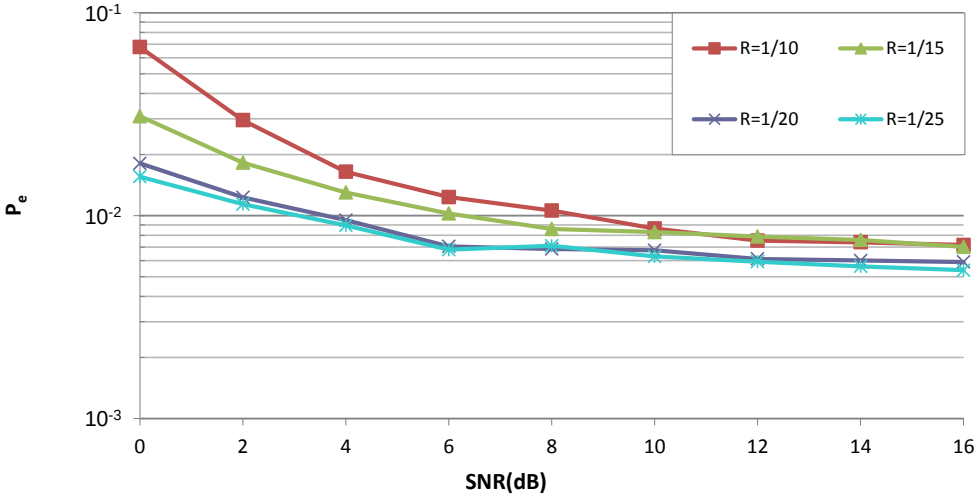


Figure 4.3: Evolution of error probability versus attack noise level (in dB) and internal convolutional code ratio R for a user group of 2^{25} and $c = 3$.

cases the opportunity of collusion with more than 3 users seems difficult. Note that in any case, if one of the traitors distributes a copy that belongs to another traitor without performing the collusion attack with her copy, the other traitor will be identified.

In this case, the turbo fingerprinting codes presented outperform both Tardos codes and Boneh-Shaw codes. By fixing the error probability the length of the turbo fingerprinting codes is at least one order of magnitude smaller than Tardos codes and three orders of magnitude smaller than Boneh-Shaw codes. It is also important to emphasize that, in our attack model, some amount of noise is included. Therefore, when comparing our codes with Tardos codes and Boneh-Shaw codes we have used the theoretical expressions in [107, 11] that did not consider the presence of noise. Note that in Table 4.2 and Table 4.3 there is no entry for Jourdas and Moulin codes [66]. Our proposed codes and the codes in [66] have the same order of magnitude in terms of length and error probability in scenarios of 3 colluders. But somehow we feel that there is no room for an explicit comparison because of the difference in construction modes. More precisely, Jourdas and Moulin use a Discrete Cosine Transformation before performing a collusion attack. Therefore, it is really difficult to evaluate if the effect of the noise added during the simulated collusion attack (which is added in the same domain of the code) is equivalent/comparable to the effect produced by the noise in [66] which is added in a different domain to the code. Finally, it is also important to point out that the tracing error probability of the turbo fingerprinting codes can be further improved. For instance, in [25] they can be used as the inner binary codes in the concatenated construction instead of Boneh-Shaw codes.

Table 4.2: Length comparison with Boneh-Shaw and Tardos contructions considering a collusion of 2 attackers (c=2).

			Lengths		
n	c	P_e	***BS***	***Tardos***	***Proposed***
8	2	0.00002	330512	6546	122
8	2	0.00001	357807	6823	692
16	2	0.00035	325356	7619	204
16	2	0.00003	426735	8602	1084
32	2	0.00043	512359	11973	348
32	2	0.00002	677276	13200	2068

Table 4.3: Length comparison with Boneh-Shaw and Tardos contructions considering a collusion of 3 attackers (c=3). *Note that in BS and Tardos constructions, the channel is considered noiseless, nevertheless, some noise is considered in the proposed construction (not considered in BS or Tardos).*

			BS	***Tardos***	***Proposed***	
n	c	P_e	***Lengths***	***Lengths***	***SNR***	***Lengths***
25	3	0.01555	1508015	19343	0 dB	1525
25	3	0.01140	1563691	19623	2 dB	1525
25	3	0.00895	1610856	19840	4 dB	1525
25	3	0.00680	1665075	20088	6 dB	1525
25	3	0.00710	1656509	20049	8 dB	1525
25	3	0.00630	1680271	20156	10 dB	1525

4.4.3 Puncturing effects on proposed codes

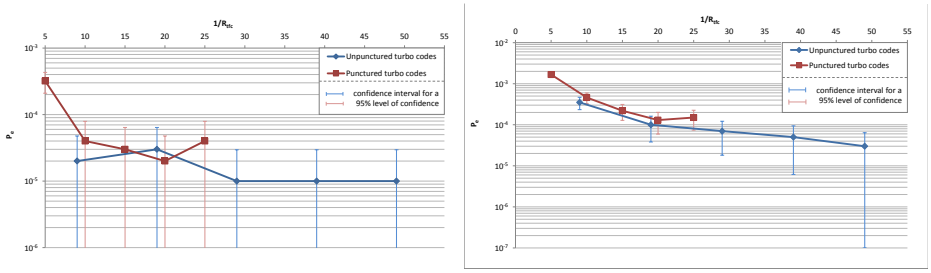
The same simulations shown in Table 4.1 have been performed again, this time adding the puncturing technique. Therefore, the simulated scenario consists again on two users performing a collusion attack and adding noise that degrades the channel to an SNR of 10 dB. But in these new simulations, the parity bits generated by the two RSC encoders of the turbo encoders (that have a constraint length $K = 7$) have been punctured.

The proposed process of puncturing consists on eliminating alternatively one of the outputs generated by the constituent RSC encoders and transmitting the parity bit of the other RSC encoder. In this way, the overall turbo fingerprinting code rate is increased because the number of parity bits transmitted is reduced by half. Therefore, the length of our codes is notably decreased.

Results are obtained for the different number of bits used to represent the $UserID$ (these values are $k = 8$, $k = 16$, $k = 32$ and $k = 64$) and different overall coding rate R_{tfc} of turbo-fingerprinting codes.

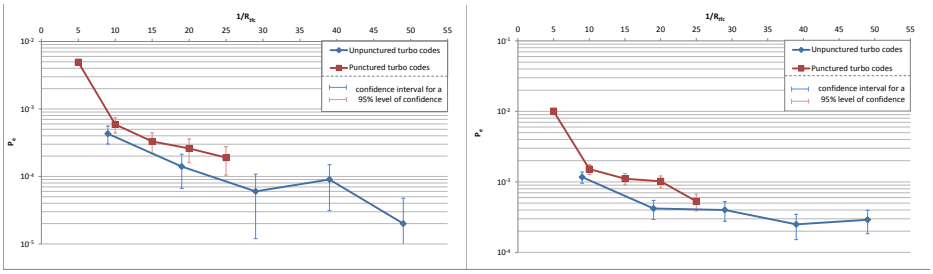
In Figure 4.4 the error probability of identifying a traitor of punctured and unpunc-

tured turbo codes is compared. As expected, the performance of the turbo codes is better when they are unpunctured than when the puncturation is used. Note that, in scenarios where the length of the code is a restriction and a large number of users is required, the puncturing process does not decrease the code performance significantly. With $R < \frac{1}{5}$ the punctured turbo code identifies colluders with a $P_e < 0.01$. And the lower the ratio, the lower the difference between the error probability of identifying a traitor of punctured and unpunctured turbo codes. Even though the code length can be adapted to smaller sizes by means of the puncturing, the simulations show that it is not an efficient solution for the studied scenario due to the kind of errors produced by the collusion attack.



(a) Error probability using 8 bits to represent the $UserID$ ($k = 8$)

(b) Error probability using 16 bits to represent the $UserID$ ($k = 16$)



(c) Error probability using 32 bits to represent the $UserID$ ($k = 32$)

(d) Error probability using 64 bits to represent the $UserID$ ($k = 64$)

Figure 4.4: Effect of puncturing the parity bits generated by the RSC encoders of a turbo encoder. The turbo fingerprinting code has two identical RSC encoders with generator polynomials F_{s_1}, F_{s_2}, F_{s_3}, F_{s_4}, F_{s_5}. Simulation results obtained using a turbo code with $c = 2$, $SNR = 10dB$ and constraint length $K = 7$.

4.4.4 On the selected algorithm and implementation details of the watermarking layer

Watermarking is a technique that consists of embedding information into data files. When using watermarking schemes, all the copies of a file contain a mark, that can be recovered at any time by a specific software.

From the detection point of view watermarking systems can be classified in two groups: blind watermarking and non-blind watermarking. In the first group of algorithms it is assumed that the original content is not needed to perform the recovery of the mark. The algorithms in the second group assume that the original content can be used by the decoder in order to improve the performance of the whole system. Since in our scenario it can be assumed that the decoder has access to the original content we have decided to use non-blind algorithms, which are faster and have fewer security issues. An extensive analysis about the problems present in blind watermarking systems is discussed in [28].

Watermarking in the frequency domain Watermarking can be applied in the frequency domain. After the transformation to the frequency domain the mark is embedded. There are different transforms we can apply, for example, the Fast Fourier transform (FFT) or the Discrete Cosine Transform (DCT). The one commonly used is the DCT, since it is proved that is the one that has a better response to the image watermarking needs. After the transformation to the frequency domain the mark is added to the signal. The inverse process transforms again the signal to the spatial domain.

In the frequency domain the watermarking process can be seen as a communication system in which the ideas that Costa presented in [24] can be applied. In this way, the original document can be seen as the communication channel and the watermark can be seen as the signal transmitted over the channel. Attacks or unintentional transformations will be modeled as noise introduced into the channel.

To embed a mark in the frequency domain we have to modify the coefficients of the transformed domain. Each frequency coefficient has a perceptual capacity. This perceptual capacity is the capacity of each coefficient to receive additional information (watermark), without introducing perceptual differences in the content. Note that the Human Visual System model (HVS) naturally works with frequencies. So working in the frequency domain helps adjusting the addition of information to the HVS sensitivity.

Secure Spread Spectrum Even though a lot of image watermarking systems exist, and some of them show better performance than secure spread spectrum, the algorithm presented in [27] by *Cox et al.* has been used in our implementation because it is easy to combine with a fingerprinting code and it offers a good protection against the typical distortions produced by video transcoding. Moreover, the computational cost of this algorithm, which is an important issue to take into account, is lower than others based on informed embedding or informed coding. In addition, as was presented in [27], the Spread Spectrum offers enough robustness against scaling, transcoding and compression, being these the main processes performed during video manipulation.

Spread Spectrum watermarking is an example of spatial embedding of watermarks. The watermark is modulated by a pseudo-noise generator in order to produce a spread spectrum signal, which is then scaled according to the required power. The modulated signal is then added to the original image to produce the watermarked image.

To detect the watermark, a high pass/edge detection or Wiener filter is applied to the watermarked image to remove irrelevant information. The output of the filter is then correlated with the modulating pseudo-noise signal used at the transmitter side and compared to a predetermined threshold for the detection of the watermark. In general, most watermarking techniques are considered a variation of the spread spectrum technique.

In [27], the mark $X = \{x_1, \cdots, x_n\}$ is embeded into $V = \{v_1, \cdots, v_n\}$ to obtain $V' = \{v'_1, \cdots, v'_n\}$. In that proposal, V is a vector which contains all DCT matrix coefficients to be marked. The three different ways for computing V' presented in [27] are

$$v'_i = v_i + \alpha x_i \tag{4.2}$$

$$v'_i = v_i(1 + \alpha x_i) \tag{4.3}$$

$$v'_i = v_i(e^{\alpha x_i}) \tag{4.4}$$

Equation 4.2 is always invertible, and 4.3 and 4.4 are invertible if $v_i \neq 0$.

At the decoder, X^* is extracted from V^* and V. In the original proposal, the similarity of X and X^* is computed to measure how similar is the extracted mark and the original mark. A minimal value of similarity is defined as a threshold T to guarantee that no false positives are present.

Modified-Cox algorithm implementation details The watermarking process performed by this sequence generator is slightly different from the explained before and it is based on Equation 4.2. Essentially, the α value takes into account the quantization process which is applied to each coefficient. Formally, the performed modification is

$$v'_i = v_i + \alpha Q_i x_i \tag{4.5}$$

where Q_i is the quantizer matrix value, α is a strength factor, and x_i is the value of the i-position of a fingerprint in polar format (that is, +1 for value 1 and -1 for value 0). Finally, the rate is the number of DCT matrices without mark between two marked matrices minus 1. In other words, when rate has the value 1, all DCT matrices are watermarked.

Usually, if there are more suitable positions to be marked than bits to be embedded, the fingerprint is repeated along all markable positions on the sequence. Because of this repetition the embedding system has a behaviour similar to a repetition code.

Finally, the extraction is done by means of comparing the original video with the received one. If the difference is greater than a given threshold, we will consider that the originally embedded value was 1. If this difference is lower than another threshold, we will consider -1 as the original value. Otherwise, 0 will be used as the original value, meaning that we assume that the value has been erased by the attacker. During the next subsections we will discuss the effect of this watermarking layer over the proposed fingerprinting codes.

4.4.5 Innocent-user framing probability versus Watermarking-to-Noise Ratio

In these simulations, two users perform a collusion attack by means of comparing their videos content. A codeword has been embedded in each MPEG2 video. A given codeword, say u^i corresponds to a given user ID, say $UserID_i$. That is, each codeword identifies a single user. As a result of the collusion attack, the attackers generate a new MPEG2 video that contains a pirate word. Moreover, as explained at Section 4.4, the colluders add additive gaussian noise to the pirate word $\mathbf{z}$. Therefore, every symbol that belongs to the pirate word $\mathbf{z}$ is modified by the noise in a different way ($\widehat{\mathbf{z}} = (\widehat{z}_1, ..., \widehat{z}_n)$).

The Watermark-to-Noise Ratio (WNR) of the marks in dB is defined as:

$$WNR = 10\log_{10}\left(\frac{E_b}{\sigma_n^2}\right) \tag{4.6}$$

where E_b is the energy per bit and σ_n^2 the noise variance. The innocent-user framing probability, that is, the error probability P_e, has been obtained for every pirate word that belongs to $\widehat{\mathbf{z}}$. These simulations have been performed with: $E_b = 20^2$, zero mean gaussian noise and different variances. Figure 4.5 illustrates how the proposed codes identify colluders with a $P_e < 0.01$ if $E_b \geq \sigma_n^2$ (i.e., if the signal level is at least equal to the noise level). Every P_e result has been obtained by means of 20.000 tests.

4.4.6 Effect of the use of a repetition code in the performance of the whole system

Since the amount of markable positions in a MPEG2 video is really large, there exist two strategies to select the code to use. On one hand, the lowest ratio code (so, the code with better performance) could be used. The problem due to the use of this code is that it will be more sensitive to cropping attacks. That is to say, the attackers crop some parts of the video and, with them, some symbols of the mark are lost. On the other hand, a shorter code could be used in conjunction with a repetition code. It is

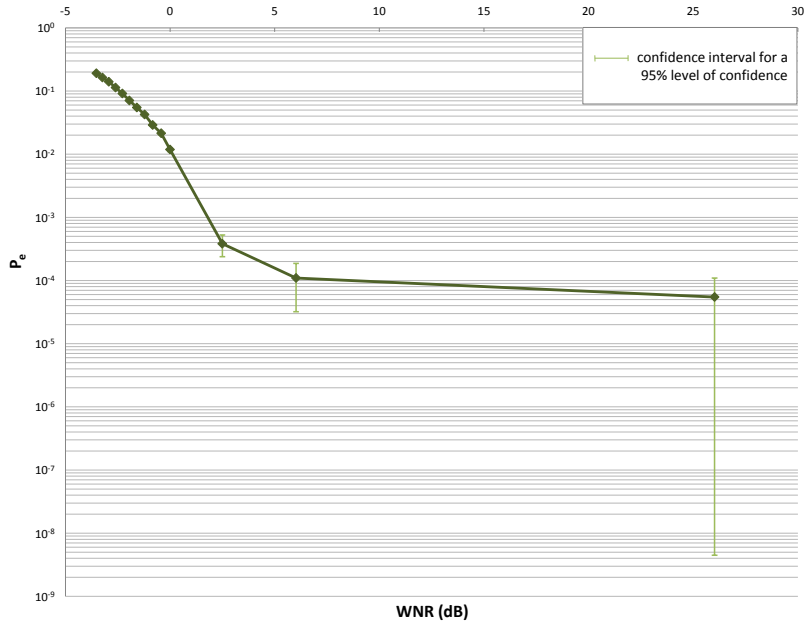


Figure 4.5: Evolution of error probability versus attack noise level (in dB), for a user group of 2^8, using a turbo fingerprinting code with $c = 2$, constraint length $K = 7$ and internal convolutional code ratio $R = \frac{1}{15}$.

clear that the individual performance of the code is worse when the overall ratio (R_{tfc}) is increased.

If u^i is defined as the codeword of the user with identifier $UserID_i$, the extended codeword $\widetilde{u}^i$ is defined as $\widetilde{u}^i = ({u^i}_1, ..., {u^i}_r)$, where ${u^i}_m = u^i$ for any $m \in \{1, \ldots, r\}$. That is to say, the encoder repeats the codeword r times.

In this simulation, two users perform a collusion attack by means of comparing their videos. A codeword has been embedded r times in each MPEG2 video (e.g. due to the length of the video used in the simulations, an extended codeword of length 412 could be embedded up to 18,295 times).

As a result of the collusion attack, the attackers generate a new MPEG2 video that contains r pirate words. So, let $\mathbf{z} = (z_1, ..., z_r)$ the set of the resultant pirate words, where z_p could be different of z_q for any $p, q \in \{1, \ldots, r\}$ depending on the collusion attack strategy. Moreover, as explained at Section 4.4, the colluders add additive Gaussian noise to $\mathbf{z}$. Therefore, every pirate word that belongs to $\mathbf{z}$ is modified by the noise in a different way, that is, $\widehat{\mathbf{z}} = (\widehat{z}_1, ..., \widehat{z}_r)$.

The WNR of the marks is of -3.5218 dB ($E_b = 20^2$ and $\sigma_n^2 = 30^2$). The value of $\sigma_n^2 = 30^2$ has no practical sense because the video would be extremely degraded with so much noise.

However, this value of σ_n^2 allows to find a greater number of iterations r required, because the error probability associated to $\sigma_n^2 = 30^2$ is high enough ($P_e = 0.1659653$).

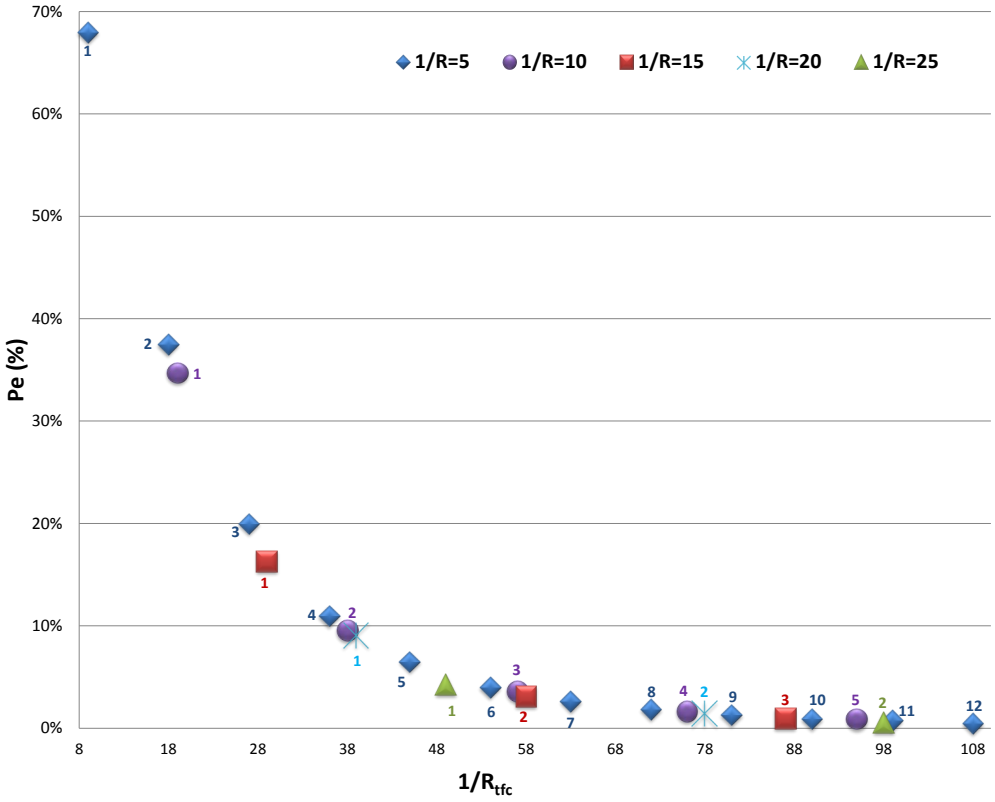


Figure 4.6: Evolution of error probability versus the number r of repetitions of the codeword of the users (r is indicated near every point in the figure), for a user group of 2^8, using different turbo fingerprinting codes with $c = 2$, constraint length $K = 7$, internal convolutional code ratios $R = \{\frac{1}{5}, \frac{1}{10}, \frac{1}{15}, \frac{1}{20}, \frac{1}{25}\}$ and WNR=−3.5218 dB.

Repetition decoding outputs the average of each output codeword belonging to $\hat{\mathbf{z}}$ as $\frac{1}{r}\sum_{k=1}^{r} \frac{u^i{}_k + u^j{}_k}{2} + n_k$ where n_k is the amount of noise added in this position. Finally, the Max-Log-MAP iterative decoding is applied to identify one of the traitors ($UserID_i$ or $UserID_j$).

As Figure 4.6 shows, a lower ratio code could be replaced by another with higher ratio using repetition (which will produce a code with similar total rate) without a big impact in the performance. Note that the simulation shown in Figure 4.6 has been done using a really high WNR as it has been explained before in this subsection.

Moreover, adding this repetition code decreases the effect of the Gaussian noise added by the colluders on the set of pirate words embedded into MPEG2 videos. Even thought, in this subsection we analyze how this performance penalty could be overcome with a repetition code.

Moreover, as Figure 4.7 illustrates, these simulations have been performed for a different number of repetitions of the codeword of the users, from $r = 1$ to $r = 15$. When the number of repetitions is larger than $r = 6$, the error probability of identi-

fying a traitor is lower than 0.001. Note that increasing r up to 15 does not decrease significantly the probability of error P_e. Therefore, the most suitable value to r seems to be $r = 7$.

4.5 Conclusions

The work presented in this chapter discusses the use of turbo codes as fingerprinting codes. Turbo fingerprinting codes based on MFD low-rate convolutional codes have been proposed as a family of fingerprinting codes secure against coalitions of size 2. It is shown by simulation that the proposed codes identify traitors with an error probability of at most $5 * 10^{-4}$ when the number of users lies between 2^8 and 2^{32}.

Moreover, when the number of colluders is $c = 3$, our system performance is also acceptable with a $P_e < 0.01$ even when the noise added by the colluders degrades the channel to an SNR of 4dB.

Finally, it is important to stress that the presented codes have an efficient decoding algorithm based on the Max-Log-MAP iterative decoding algorithm. As a future research, it would be interesting to study a list-decoding algorithm suitable for turbo-decoding to obtain the two most likely colluders.

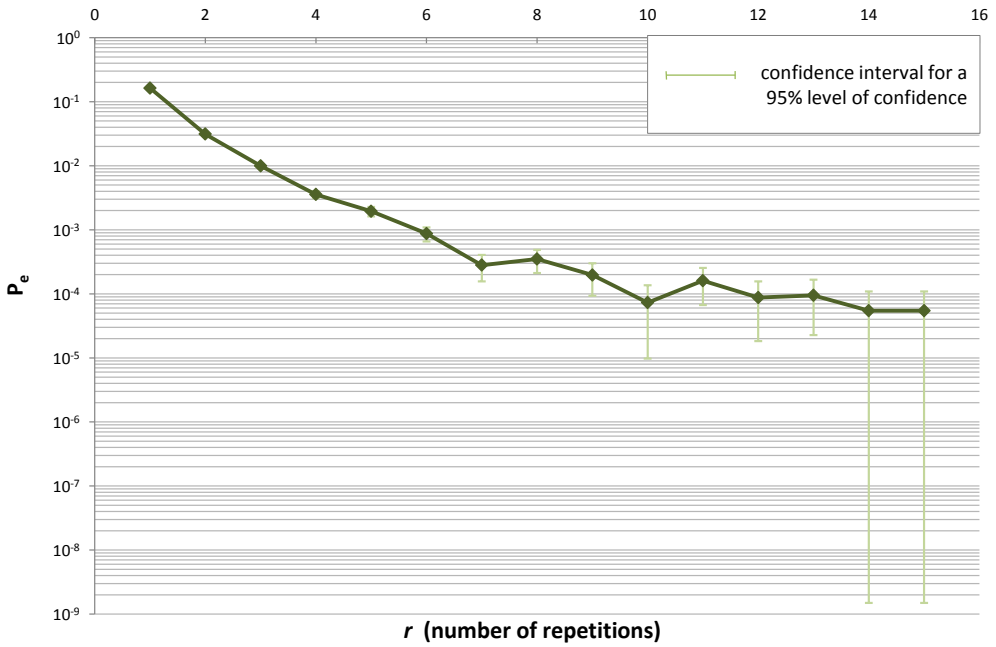


Figure 4.7: Evolution of error probability versus the number of repetitions of the codeword of the users (r), for a user group of 2^8, using a turbo fingerprinting code with $c = 2$, constraint length $K = 7$, internal convolutional code ratio $R = \frac{1}{15}$ and WNR=−3.5218 dB.

Part III

Contributions related to secure e-commerce of multimedia content

CHAPTER 5

TRAITOR TRACING OVER YOUTUBE VIDEO SERVICE - PROOF OF CONCEPT

The development explained in this chapter proves that is possible to trace dishonest users who upload videos with sensitive content to the YouTube service. To achieve tracing these traitor users, fingerprint marks are embedded by a watermarking algorithm into each copy of the video before distributing it. Our experiments show that if the watermarking algorithm is carefully configured and the fingerprints are correctly chosen, the traitor, or a member of a set of traitors who have performed a collusion attack, can be found from a pirate video uploaded to the YouTube service.

5.1 Introduction

The distribution and playback of digital images and other multimedia products is an easy task due to the digital nature of the content. The service of broadcasting videos to lots of users by means of web services has increased in number and quality during recent years. Nowadays it is extremely easy for a user to upload a video to any of the existing video broadcasting webs and distribute a link to this video. One of the most used broadcast services is the well-known YouTube broadcast service. Currently the copyright management in the YouTube service is a reactive process, that is, when a distributor finds some copyrighted work in YouTube, YouTube is informed by this distributor and the copyrighted media is removed from YouTube service, by the YouTube administrators. At the early stages of the service, when YouTube banned a video, usually, this video was reuploaded and the removal process needed to be repeated, in other words, the distributors needed to find the new copy, inform YouTube,... Nowadays,

YouTube tries to identify copies of already banned videos in order to simplify this process. These techniques are also known by the name of fingerprinting but not in the sense of the fingerprinting techniques that are discussed in this chapter. Some new contribution from Google researchers are published in [4, 26, 5].

On the other hand, some new Digital Right Management (DRM) systems are included into Flash Player, the DRM protections it offers, only aims at the copyright distribution, for instance, playing a video a limited number of times. The work presented in this chapter is addressed to trace who has upload a video to YouTube.

Achieving satisfactory copyright protection has become a challenging problem for the research community. The mechanisms used in protecting intellectual property from unauthorised copying can be classified into two groups: copy prevention mechanisms and copy detection mechanisms. The failure of copy prevention schemes, such as the DVD copy prevention system, has shifted many research efforts towards the search of feasible copy detection mechanisms. These efforts have brought the development of new proposals of copy detection systems for many different types of digital content.

An important number of these new proposals can be grouped under the name of watermarking. The watermarking technique consists of embedding a set of marks in each copy of the digital content. The more important issues that watermarking algorithms have to take into account are the processes in which the digital document can be involved and how the mark embedded into this digital document is affected by them. One of the seminal paper about watermarking was presented by Cox *et al.*. In [27], the authors present a novel image watermarking algorithm called Secure Spread Spectrum which seems to be able to resist typical image processing and offers a correct level of robustness.

Since the embedded set of marks is the same for all copies, watermarking schemes ensure intellectual property protection but fail to protect distribution rights. In the proposed scenario, the protection against distribution rights is also required, then the concept of watermarking needs to be extended further off. The functionality of watermarking can be extended by allowing the embedded set of marks to be different in each copy of the object. This true original idea is called fingerprinting because of its analogy to human fingerprints. The concept of fingerprinting was introduced by Wagner in [116] as a method to protect intellectual property in multimedia contents. The fingerprinting technique consists in making the copies of a digital object unique by embedding a different set of marks in each copy. Having unique copies of an object clearly rules out plain redistribution, but still a coalition of dishonest users can collude. A collusion attack consist in comparing the copies of the coalition members and by changing the marks where their copies differ, they create a pirate copy that tries to disguise their identities. Observe that in this situation it is possible for the attackers to

frame an innocent user. Thus, the fingerprinting problem consists in finding, for each copy of the object, the right set of marks that help to prevent collusion attacks.

The chapter is organised as follows. In section 5.2 the working scenario is briefly described. Section 5.3 provides an overview about watermarking techniques. Section 5.4 presents the fingerprinting codes in general and describes how to construct and decode one particular family of such codes. In section 5.5, the YouTube video service is introduced. Section 5.6 discusses our implementation. In section 5.7, the obtained results are explained. Finally, some conclusions are given in section 5.8.

5.2 Scenario description

Figure 5.1 shows the considered scenario. Suppose that a confidential video must be shown to a reduced group of people. As an example imagine a surveillance camera video footage which has to be presented in a criminal proceeding. Some copies of this video are distributed to the judge, members of the jury, etc... and one of the copies is uploaded to Youtube. If all copies are identical, that is, without any protection against redistribution, detecting who has uploaded it is not feasible.

The whole scenario starts with the distribution step. As has been shown before, the original video is watermarked with different fingerprints to generate several fingerprinted copies. Each one of these copies is distributed to the users. Then, users A and B collude in order to avoid the redistribution protection tracing, and they generate a new copy of the video with a false fingerprint which is not exactly the fingerprint of neither A and B. Finally, the traitors upload the colluded video to YouTube service.

When a video is uploaded to YouTube, their servers performs some processing to this video. Therefore, compression and scaling processes have to be taken into account because both of them can distort the mark causing a fingerprinting information loss. That is, the original video must be adapted to obtain the correct performance and to minimize the YouTube service processing effect. In our simulations we use videos in MPEG-2 format with resolutions and sizes near to the video properties of the YouTube service so; the distortion produced by the YouTube service processing is lower than if high resolutions and sizes will be used so, at tracing time, no pre-processing is required and our simulations are faster. Nevertheless, in a real environment, that is, with videos in high resolutions, some post-processing will be needed and, it is possible that more robust watermarking algorithm must be used. However, in our study we assume that the original videos are in the appropriate resolution and size (this is more deeply explained in section 5.7).

The next step is to retrieve the video from Youtube and transform it in order to be able to use it as input for our recovery mark system because the watermarking algorithm which is used is non-blind and it extracts the mark from the watermarked video

by the comparison with the original unmarked video. When the fingerprint have been extracted, it is decoded by a fingerprinting decoder thus identifying at least one of the traitors.

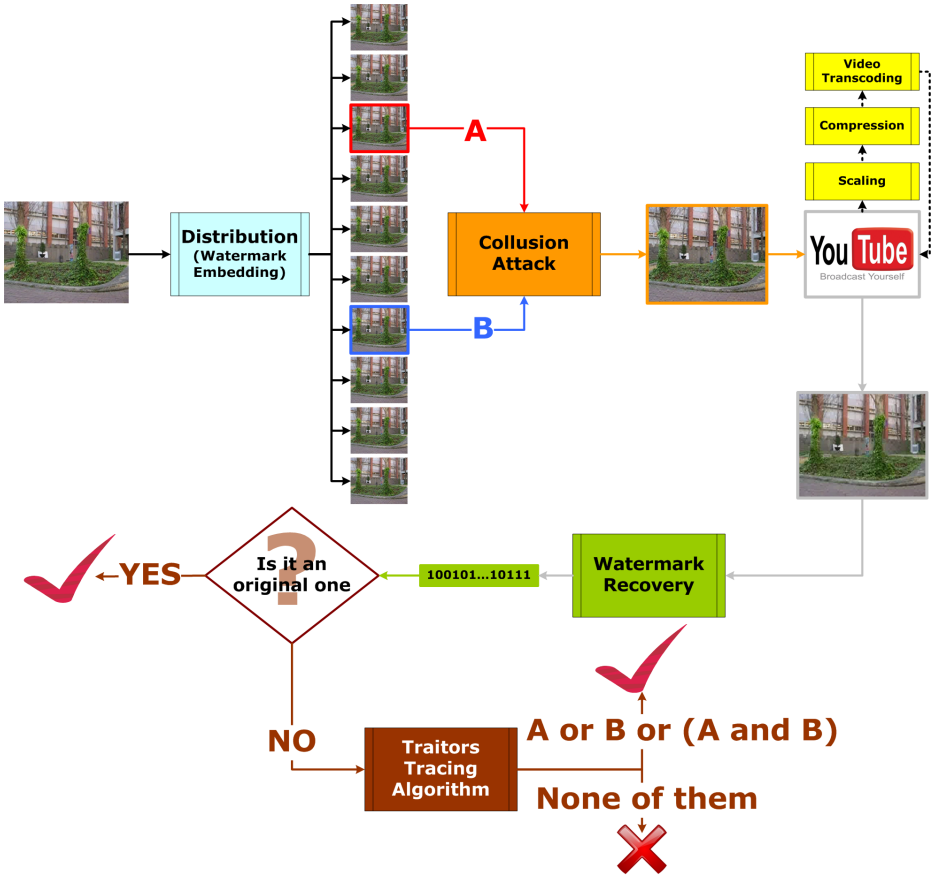


Figure 5.1: Overall system

5.3 Watermarking Layer

Watermarking is the technique of embedding information into data files. It is necessary to make a little distinction between cryptography and watermarking. These two techniques have different purposes. Cryptography ciphers a message with the intention of being read just by a specific user. Therefore this particular user decrypts the ciphered message and gains access to it. After decrypting the message the retransmission of the same message can be done in clear. In watermarking schemes, in every copy of the file, in each retransmission, the embedded information is present, and at any time it can be recovered by some software suitably written.

From the detection point of view, the watermarking systems can be classified in two groups: blind watermarking and non-blind watermarking. The first one assumes

that, at decoding time, the original document, which is the content without any watermark embedded, is not available. The second kind of systems assume that the original content can be used by the decoder in order to improve the performance of the whole system. Because usually, non-blind algorithms have better performance and need to deal with less security issues and because in our scenario it can be assumed that the decoder has access to the original content; non-blind watermarking algorithm seemed the best choice for us. An extensive analysis about the problems presented by blind watermarking systems is discussed in [28].

5.3.1 Watermarking in the frequency domain

Watermarking can be applied in the frequency domain. After the transformation to frequency domain the mark is inserted. There are different ways for transforming the domain, for example, Fast Fourier transform (FFT) or Discrete Cosine Transform (DCT). The most used is DCT, since it is proved that is the one that has a better response to the watermarking needs. After the transformation to the frequency domain the mark is added to the signal. The inverse process transforms again the signal to the spatial domain.

In frequency domain the watermarking process can be seen as a communication system in which the ideas of Costa presented in [24] can be applied. In this way, the original document can be seen as the communication channel and the watermark can be seen as the signal transmitted over the channel. Attacks or unintentional transformations will be seen as noise introduced into the channel.

There are many advantages in placing the watermark in the frequency domain. As it also happens in the spatial domain the different frequency values can be modified. Each frequency coefficient has a perceptual capacity. This perceptual capacity is the capacity of each coefficient to receive additional information (watermark), without introducing perceptual differences in the image or audio. Note that the Human Visual System model (HVS) naturally works with frequencies. So working in the frequency domain helps adjusting the addition of information together with the HVS sensitivity.

One of the traditional watermarking techniques in frequency domain is the spread spectrum [27]. Spread spectrum techniques have been widely used in watermarking applications to embed small amounts of bits given their robustness against different intentional an unintentional attacks. Quantization Index Modulation (QIM) based methods have become more popular for their higher data capacity and their property of presenting zero probability of error decoding error for certain amplitude bounded-attacks [15]. Nevertheless they may fail in high distortion contexts as in our proposed scenario. In recent years, a new watermarking model with large data payload has appeared. This scheme is based on the parallel use of two different techniques: Informed

Coding [80] and Informed Embedding [81]. The same approach is used in both techniques and it is based on the fact that the transmitter knows the original content (cover work). A deeper study about the joint use of these two techniques can be found in [82]. The main objectives of Informed Embedding technique are to adapt every watermark to the cover work maintaining a relationship between the watermarks desired robustness and the distortion effect produced in the image. On the other hand, the basis of Informed Coding is to perform an intelligent message codification depending on the cover work. One problem of these techniques is excessive time consuming. In [67], an optimization over this issue is proposed by means of the use of Hadamard matrix. However, the use of Informed Embedding or Informed Coding with fingerprinting codes presents important problems. In [TBFS07], some improvements in order to adapt these codes to fingerprinting systems are discussed. A more general and broader discussion on data hiding codes is provided in [84].

5.3.2 Secure Spread Spectrum

Even though there exists lots of image watermarking systems and some of them show better performance than secure spread spectrum, this algorithm presented in [27] by *Cox et al.* has been used in our implementation because it is easy to combine with a fingerprinting code and it offers a good protection against the distortions produced by the YouTube service. Moreover, the time consuming of this algorithm is lower than others based on informed embedding or informed coding so, as during our simulations we needed to send hundreds of marked videos to the YouTube service, the time consuming was an important restriction to take into account. In addition, as was presented in [27], the Spread Spectrum offers enough performance against scaling, transcoding and compression, which are the main processes performed by the YouTube service.

Spread Spectrum watermarking is an example of spatial embedding of watermarks. The watermark is modulated by a pseudo-noise generator in order to produce a spread spectrum signal, which is then scaled according to the required power. The modulated signal is then added to the original image to produce the watermarked image.

To detect the watermark, a high pass/edge detection or Wiener filter is applied to the watermarked image to remove irrelevant information. The output of the filter is then correlated with the modulating pseudo-noise signal used at the transmitter side and compared to a predetermined threshold for the detection of the watermark. In general, most watermarking techniques are considered a variation of the spread spectrum technique. The whole system is briefly presented in figure 5.2.

In [27], the mark $X = \{x_1, \cdots, x_n\}$ is embeded into $V = \{v_1, \cdots, v_n\}$ to obtain $V' =$

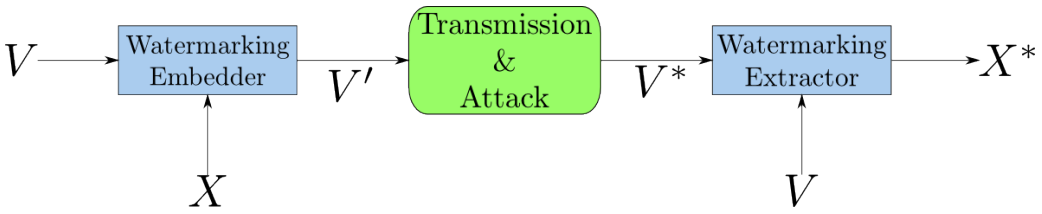


Figure 5.2: Watermarking process schema

$\{v'_1, \cdots, v'_n\}$. In that proposal, V is a vector which contains all DCT matrix coefficients to be marked. Three natural formulae for computing V' presented in [27] are

$$v'_i = v_i + \alpha x_i \tag{5.1}$$

$$v'_i = v_i(1 + \alpha x_i) \tag{5.2}$$

$$v'_i = v_i(e^{\alpha x_i}) \tag{5.3}$$

Equation 5.1 is always invertible, and 5.2 and 5.3 are invertible if $v_i \neq 0$.

At the decoder, X^* is extracted from V^* and V. In the original proposal, the similarity of X and X^* is computed to measure how similar is the extracted mark and the original mark. A minimal value of similarity is defined as a threshold T to guarantee that no false positives are present.

5.4 Fingerprinting Layer

As it has been introduced before, the fingerprinting problem consists in finding, for each copy of the object, the right set of marks that help to prevent collusion attacks. The codes that can resist to collusion attacks are called collusion secure codes. The construction of collusion secure codes was first addressed in [12]. In that paper, Boneh and Shaw obtain $(c > 1)$-secure codes, which are capable of identifying a guilty user in a coalition of at most c users with a probability ϵ of failing to do so. The construction composes an inner binary code with an outer random code. Therefore, the identification algorithm involves the decoding of a random code, that is known to be a NP-hard problem [6]. Moreover, the length of the code is considerably large for small error probabilities and a large number of users.

To reduce the decoding complexity, Barg, Blakley and Kabatiansky in [6] used algebraic-geometric codes to construct fingerprinting codes. In this way, their system reduces the decoding complexity to $O(poly(n))$ for a code length n and only 2 traitors. In [38], Fernandez and Soriano constructed a 2-secure fingerprinting code by concatenating an inner $(2,2)$-separating codes with an outer IPP code (a code with the Identifiable Parent Property), and also with decoding complexity $O(poly(n))$. In [107], Tardos presents a codes which have length $O(c^2 log(n/\epsilon))$ and are ϵ-secure against c

pirates over n users. These codes represented an important improvement compared to the Boneh and Shaw proposal.

Another point of view about fingerprinting codes applied to media has been presented recently by He and Wu. Their proposals take advantage of joint coding and embedding that is, designing fingerprinting codes taking into account the content in which will be embedded. In [57], a system, called GRACE, which assumes a small number of users is discussed. These users are separated in groups of orthogonal fingerprints, that is, every fingerprint is orthogonal inside his group. In [56], this system is improved by means of the use of Reed-Solomon codes and it can accommodate more than ten million users resisting collusions performed by hundreds of users.

In our implementation, the code and algorithms presented in [38] has been used as fingerprinting code because, in our scenario, there are a small number of users and the performance, from the time consuming and tracing capacity point of view, is good enough.

5.4.1 Background on coding theory

A subset C of a vector space $\mathbf{F}_q^n$ is called a *c*ode. The set of scalars $\mathbf{F}_q$ is called the *c*ode alphabet. A vector in $\mathbf{F}_q^n$ is called a *w*ord and the elements of C are called *c*odewords, each codeword is of the form $\mathbf{x} = (x_1, \ldots, x_n)$ where $x_i \in \mathbf{F}_q$, $1 \leq i \leq n$.

The number of nonzero coordinates in $\mathbf{x}$ is called the *w*eight of $\mathbf{x}$ and is commonly denoted by $w(\mathbf{x})$. The *H*amming distance $\mathbf{d}(\mathbf{a}, \mathbf{b})$ between two words $\mathbf{a}, \mathbf{b} \in \mathbf{F}_q^n$ is the number of positions where $\mathbf{a}$ and $\mathbf{b}$ differ. The distance between a word $\mathbf{a}$ and a subset of words $U \subset \mathbf{F}_q^n$ is defined as $\mathbf{d}(\mathbf{a}, U) := \min_{\mathbf{u} \in U} \mathbf{d}(\mathbf{a}, \mathbf{u})$. The *m*inimum distance of C, denoted by d, is defined as the smallest distance between two different codewords.

A code C is a *l*inear code if it forms a subspace of $\mathbf{F}_q^n$. A code with length n, dimension k and minimum distance d is denoted as a $[n, k, d]$-code.

If we take a set of n distinct elements $P = \{v_1, \ldots, v_n\} \subseteq \mathbf{F}_q$, then a *R*eed-Solomon code of length n and dimension k, consists of all codewords of the form $(f(v_1), \ldots, f(v_n))$ where f takes the value of all polynomials of degree less than k in $\mathbf{F}_q[x]$:

$$\mathrm{RS}(P, k) = \{(f(v_1), \ldots, f(v_n)) \mid f \in \mathbf{F}_q[x] \wedge \deg(f) < k\}$$

A *s*implex code or *d*ual binary Hamming code S_r, is a $[2^r - 1, r, 2^{r-1}]$ code, consisting of $\mathbf{0}$ and $2^r - 1$ codewords of weight 2^{r-1}, with every pair of codewords the same distance apart.

For any two words $\mathbf{a}$, $\mathbf{b}$ in $\mathbf{F}_q^n$ we define the set of *d*escendants $D(\mathbf{a}, \mathbf{b})$ as

$$D(\mathbf{a}, \mathbf{b}) := \{\mathbf{x} \in \mathbf{F}_q^n : x_i \in \{a_i, b_i\}, 1 \leq i \leq n\}.$$

One can see that among the set of descendants of $\mathbf{a}$ and $\mathbf{b}$, there are $\mathbf{a}$ and $\mathbf{b}$ themselves.

For a code C, the *d*escendant code C^* is defined as:

$$C^* := \bigcup_{\mathbf{a}\in C, \mathbf{b}\in C} D(\mathbf{a},\mathbf{b}).$$

5.4.2 Construction of a Concatenated Fingerprinting Code

The idea of using code concatenation in fingerprinting schemes to construct shorter codes, was presented earlier in [12].

A concatenated code is the combination of an *i*nner $[n_i, k_i, d_i]$ q_i-ary code ($q_i \geq 2$) with an *o*utter $[n_o, k_o, d_o]$ code over $\mathbf{F}_{q_i^{k_i}}$. The combination consists in mapping the codewords of the inner code to the elements of $\mathbf{F}_{q_i^{k_i}}$, that results in a q_i-ary code of length $n_i n_o$ and dimension $k_i k_o$. Note that the size of the concatenated code is the same as the size of the outter code.

In [31], dual binary Hamming codes are proposed as fingerprinting codes. One of the major drawbacks of that scheme is that the number of codewords grows linearly with the length of the code. To overcome this situation in [38], code concatenation is used, and combine a dual binary Hamming code with an IPP Reed-Solomon code.

So, to construct a $[n(2^r-1), r\lceil n/4\rceil]$ binary fingerprinting code $\mathscr{C}$, they propose:

- as inner code, a $[2^r-1, r, 2^{r-1}]$ dual binary Hamming code S_r,
- as outter code, a $[n, \lceil n/4\rceil, n-\lceil n/4\rceil+1]$ IPP Reed-Solomon code over $\mathbf{F}_{2^r}$,
- together with a mapping $\phi : \mathbf{F}_{2^r} \to S_r$.

The codewords of $\mathscr{C}$ are obtained as follows, take a codeword $\mathbf{x} = (x_1, \ldots, x_n)$ from the Reed-Solomon code and compute $\mathbf{y}_i = \phi(x_i),\ 1 \leq i \leq n$. The concatenation of the $\mathbf{y}_i$'s forms a codeword $\mathbf{y} \in \mathscr{C}$, where,

$$\mathbf{y} = (\mathbf{y}_1, \ldots, \mathbf{y}_n) \text{ such that } \mathbf{y}_i = \phi(x_i)$$

5.4.3 Overview of the Fingerprinting Concatenated Decoding Algorithm

The decoding is done in two stages. First, we decode the inner code to obtain an n-tuple of sets of codewords. Then, with this n-tuple of sets we construct a reliability matrix that is used to decode the outter code.

Suppose we want to decode the following fingerprint:

$$\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n)$$

The inner decoding consists in the decoding of each subword $\mathbf{y}_i$ using the Simplified Chase Algorithm. The output will be a single codeword $\{\mathbf{h}_1\}$, a pair of codewords $\{\mathbf{h}_1, \mathbf{h}_2\}$ or three codewords $\{\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3\}$.

Then, for $i = 1,\ldots,n$ we use the mapping $\phi(s_m) = \mathbf{h}_m$ to obtain the set $S_i^{(j)} = \{s_{i_1},\ldots,s_{i_j}\}$, where the superscript $j \in \{1,2,3\}$ indicates the cardinality of the set. Note that the elements of the $S_i^{(j)}$'s are symbols from $\mathbf{F}_{2^r}$. We denote by $\mathscr{S}^{(1)}$ the set of the $S^{(1)}$'s, by $\mathscr{S}^{(2)}$ the set of the $S^{(2)}$'s and by $\mathscr{S}^{(3)}$ the set of the $S^{(3)}$'s.

We also define the n-tuple of sets $\mathscr{S} = (S_1^{(j)},\ldots, S_n^{(j)})$, that is used to construct a reliability matrix. With this matrix we run the Koetter-Vardy algorithm obtaining a list U of potential parents. If d_o denotes the minimum distance of the outter code, then to extract the positive parents out of the list U, we use the following statements:

- If $(|\mathscr{S}^{(1)}| + |\mathscr{S}^{(2)}|) > 4(n - d_o)$, then by Theorem 3 in [38], at least one of the parents is identified with probability 1.
- If $|\mathscr{S}^{(2)}| > 2(n - d_o)$, then by Theorem 3 in [38], both parents are identified with probability 1.
- If $(|\mathscr{S}^{(1)}| + |\mathscr{S}^{(2)}|) \leq 4(n - d_o)$, then define $U_3 = \{\mathbf{u} \in U : u_p \in S_p^{(3)},\ \forall\ S_p^{(3)} \in \mathscr{S}\}$. The only cases of positive identification are:
  - For any $S_p^{(2)} \in \mathscr{S}$, where $S_p^{(2)} = \{s_{p_1}, s_{p_2}\}$, if there are two and only two codewords $\{\mathbf{u}^1, \mathbf{u}^2\} \in U_3$, such that $u_p^1 = s_{p_1}$ and $u_p^2 = s_{p_2}$, then codewords $\mathbf{u}^1$ and $\mathbf{u}^2$ can be identified as positive parents.
  - For any $S_p^{(1)} \in \mathscr{S}$, where $S_p^{(1)} = \{s_{p_1}\}$, if there is one and only one codeword $\mathbf{u} \in U_3$, such that $u_p = s_{p_1}$, then codeword $\mathbf{u}$ can be identified as a positive parent.

5.5 YouTube Broadcast video service

YouTube is a video sharing website that offers to their users the following features:

- **Video embedding:** The uploaded videos can be linked by users in other websites in order to be seen by people who visit this other website.
- **Public or private videos:** When registered users upload video, they can choose if this video can be broadcasted to anyone that connects to YouTube or only to the people who the user authorizes.
- **Subscriptions:** YouTube offers the option to syndicate to their favorite users' new videos like in RSS (Really Simple Syndication).
- **Quick Capture:** YouTube website facilitates to their users a webcam and Flash software the option of instantly record video and upload it in real time rather than having to pre-record and then upload the video.

YouTube has been growing in the number of users and video since its appearance in February 2005. Few statistics are publicly available regarding the number of videos on YouTube. However, in July 2006, the company revealed that more than 100 million videos were being watched every day, and 2.5 billion videos were watched in June 2006. 50,000 videos were being added per day in May 2006, and this increased to 65,000 by July. In January 2008 alone, nearly 79 million users had made over 3 billion video views. It is estimated that in 2007, YouTube consumed as much bandwidth as the entire Internet in 2000, and that around ten hours of video are uploaded every minute.

5.5.1 Technical notes

One key issue in our proposal is the video technologies that YouTube use and how YouTube process the upload videos. As it seems logical the videos are compressed in order to offer a reasonable broadcasting quality. Basically, YouTube accepts videos of a wide variety (MPEG 1, 2 and 4, Windows Media Video, Audio Video Interleave and QuickTime File format for instance). The users can play YouTube video by means of Adobe Flash Player which is distributed as a plugin for web browsers. The standard video format from YouTube is a Flash Video which is a proprietary file format that contains a variant of H.263 video standard known as Sorenson Spark video codec [78]. All uploaded videos are converted to 314 kbit/s as a bit rate with a variable frame rate depending on uploaded video and are scaled to 320 pixels wide by 240 pixels high. Some uploaded videos are also available in a better video definition since March 2008 but it is YouTube who decides if a specific video is converted to this better quality.

The YouTube also provides to external programmers a set of tools in order to facilitate the interaction of the YouTube website with other developments. This tools have been used to speed up our implementation.

5.6 Our implementation

The aim of this work is to implement a proof of concept that shows how a successful traitor tracing can be done in video content that has been uploaded to the YouTube service. There are two different workflows to deal with: pirate copy generation and traitor tracing process. The first one is summarized in figure 5.3 and consists in the following steps:

1. A seller, who is the owner of the content-distribution rights, uploads a copyrighted content to a ContentProvider.

2. A buyer requires a specific content from the ContentProvider.

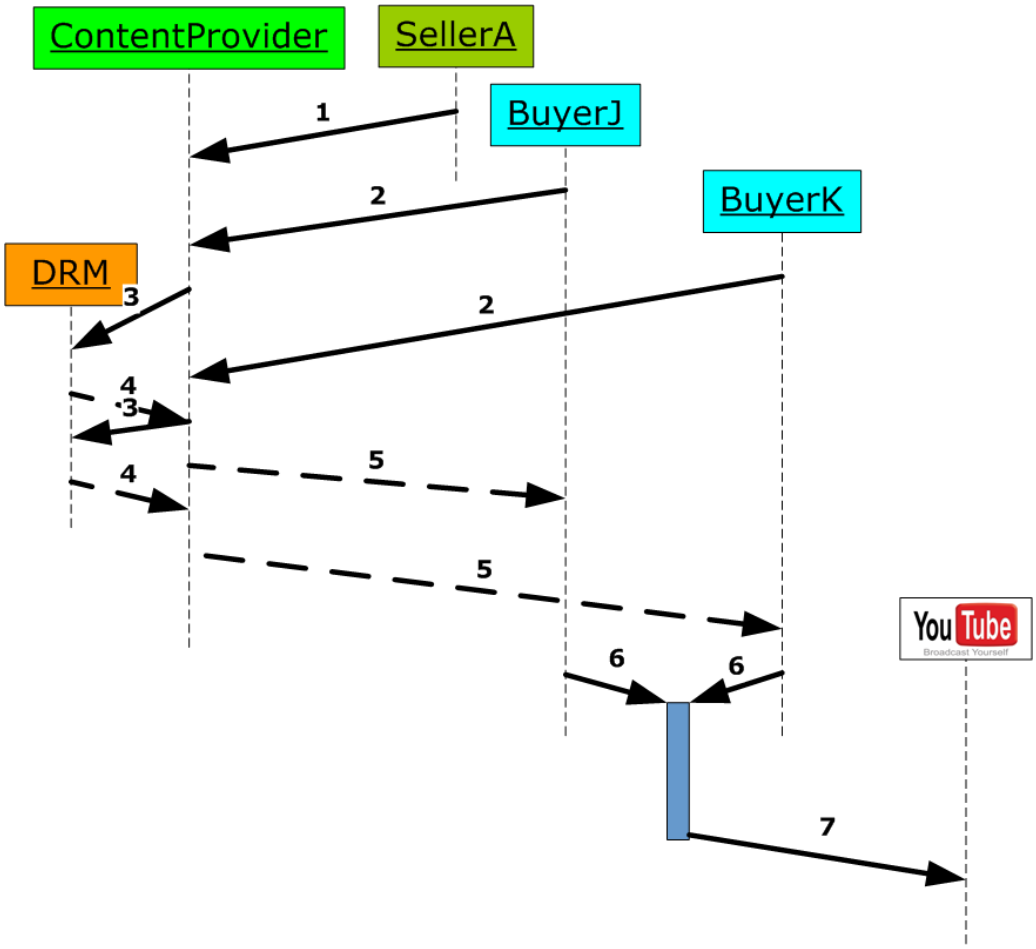


Figure 5.3: Collusion attack process.

3. The ContentProvider makes a request to the DRM provider, of the marked (with the fingerprint embedded) copy of the content that will be assigned to this BuyerJ.

4. The DRM provider sends to the ContentProvider the marked copy in which the fingerprinting that identifies BuyerJ has been embedded.

5. The marked copy is sent to the buyer.

6. Two buyers that have the same content (with different fingerprints embedded) decide to collude in order to generate a pirate copy.

7. The pirate copy is uploaded to the YouTube service.

The second workflow is shown in figure 5.4 and consists of the following steps:

1. A seller searches for a specific video in the YouTube service.

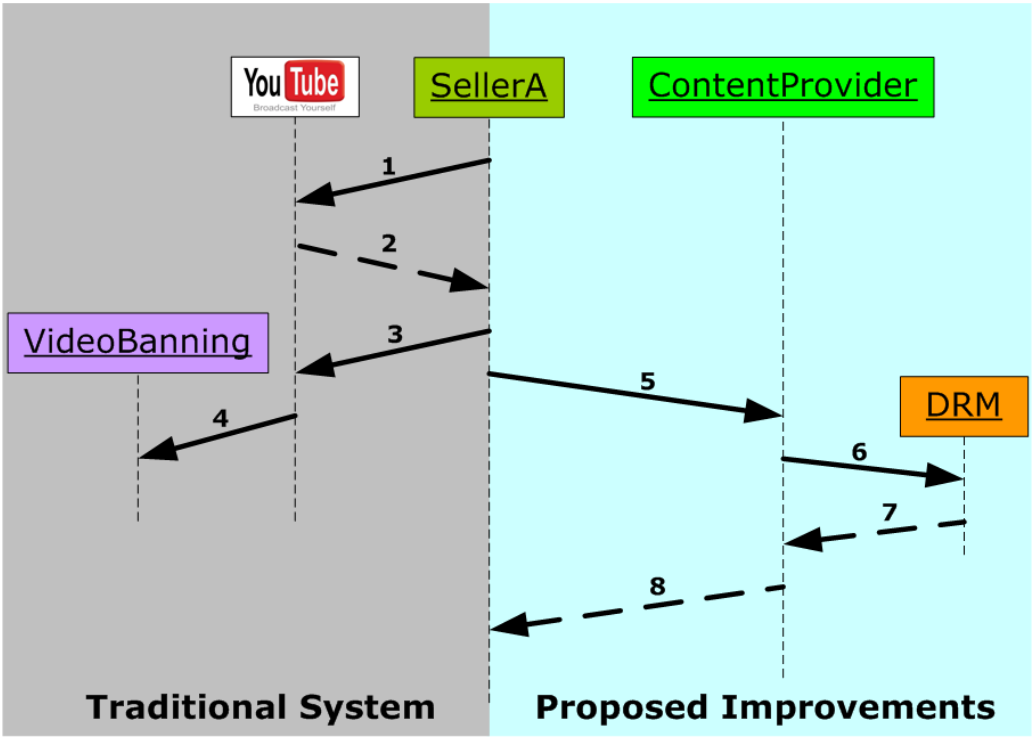


Figure 5.4: Traitors tracing workflow.

2. We assume that the search is successfull.

3. If the content is copyrighted by this seller, he could request to the YouTube administrators a banning of this content.

4. In a normal situation the YouTube administrators will ban the content so, the content will no longer be available in the Youtube service.

5. The seller sends this content to the ContentProvider in order to discover who was the user (or users) that uploaded the content to the YouTube service.

6. The ContentProvider asks the DRM provider about the identity of the buyers of this content.

7. The DRM provider extracts the mark and traces the buyers.

8. The ContentProvider sends the identity of this traitor (or traitors) and also the evidences of guilt to the seller in order to start the punishment.

Note that the first four steps are usually done in the current Youtube service. Our contribution proves that it is possible to also identify the buyers who have uploaded copyrighted contents to YouTube.

In order to perform our simulations, there are some particular stages which must be briefly described. These stages are:

1. Video acquisition: In our environment, the videos are acquired with a commonly digital camera with recording video option. The camera used has 3.2 Mpixels of resolution and the output is a file in AVI (Audio Video Interleave).

2. First transcoding process: This AVI file is transcoded to a MPEG 2 video, so the watermarking algorithm works in a DCT domain and MPEG 2 also works in this domain.

3. Fingerprint embedding: Fingerprinting embedding uses part of the code implemented in the previous work presented in [SFS+05].

4. Collusion attack: Two of the generated copies are colluded to generate a new pirate copy.

5. YouTube upload: The pirate copy is uploaded to YouTube and YouTube process as a other normal video. Finally the video is available by all users who visit YouTube website.

6. YouTube download: The pirate video is found in YouTube and it is download as a Flash video.

7. Second transcoding process: The Flash video is encoded to a MPEG 2 in a way that is as similar as possible to the generated videos.

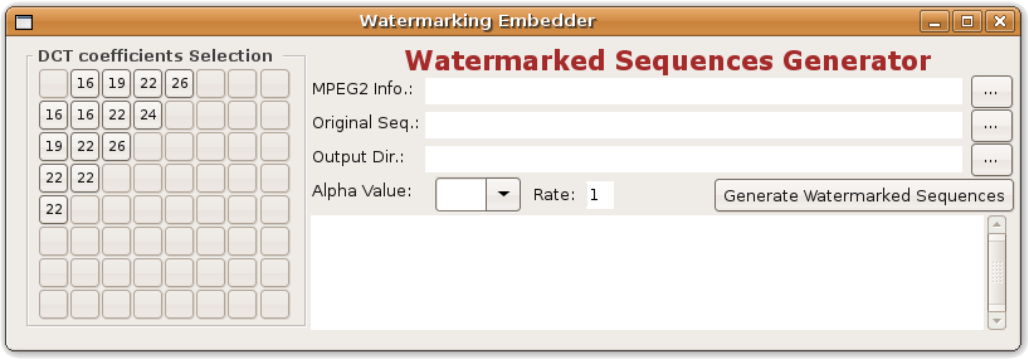
8. Fingerprint recovery: A distorted version of the fingerprint is retrieved from the pirate video.

9. Traitor tracing: Finally the traitor tracing algorithm is executed over the retrieved fingerprint.

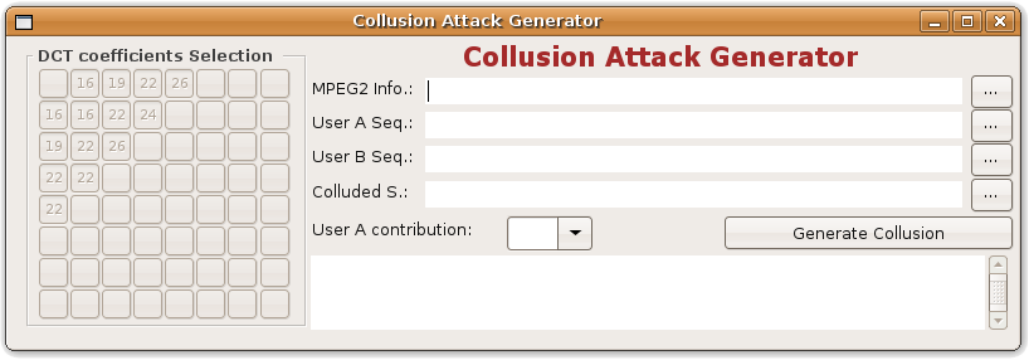
Other aspects of our implementation are explained in the following sub-sections.

5.6.1 Sequence generator

The sequence generator (fig. 5.5a) is the application that generates the fingerprints for all users and embeds it into the original sequence to produce the watermarked sequences. In our simulations, 10 users are randomly chosen between the available 2^{10} users, and our software creates one marked sequence for each of them. Some values are requested by the graphical interface. Basically, the original sequence to be protected, the output directory and the directory that contains some information about MPEG-2.



(a) Sequence generator



(b) Collusion Attack generator

(c) Traitor Tracing

Figure 5.5: Tools developed to perform our proof of concept.

The watermarking process performed by this sequence generator is a slightly different from the explained in section 5.3.2 and it is based in Equation 5.1. Essentially, the α value takes into account the quantization process which is applied in each coefficient. Formally, the performed modification is

$$v_i' = v_i + \alpha Q_i x_i \tag{5.4}$$

where Q_i is the quantizer matrix value (which is also showed in the figure 5.5a), α is a strength factor, and x_i is the mark in polar format, that is +1 for value 1 and -1 for

value 0. Finally, the rate is the number of DCT matrix without mark are beetwen two marked matrix minus 1, in other words, when rate has the value 1, all DCT matrix are watermarked.

As usually there are more positions suitable to be marked than bits to insert, the mark is repeated during all markable positions along the sequence. In this way, this embedding system has a behavior similiar to a repetition code.

5.6.2 Collusion attack generator

As has been explained before, the attack that fingerprinting tries to avoid is the collusion attack. In this attack, the traitors (2 in our case) compare their copies of the same content and look for different positions. After localizing these positions, the traitors generate a new copy in which the values of these positions are chosen randomly between the different values of their original copies.

Our implementation (fig. 5.5b) asks for the sequence of the 2 traitors, the place to put the colluded sequence, and the contribution of every user. The worst case, from the fingerprinting algorithm point of view, is that every traitor has the same contribution.

On the other hand, as the attackers do not know which DCT coefficients were used to embed the watermark, they use all 14 possible DCT coefficients to collude.

5.6.3 Traitor Tracing

After localizing a pirate copy of a video in YouTube, we retrieve it and recode it to the original format, resolution and bit rate; the the traitor tracing algorithm can be applied. Our implementation (fig. 5.5c) asks for the colluded sequence and the original sequence. Take into account that the implemented watermarking algorithm is a non-blind algorithm, that is, the original sequence is needed to recover the embedded watermark.

This algorithm will output at least one traitor if the number of errors added by the watermarking layer is close to 0.

5.6.4 External tools

Some external tools have been used in order to perform some processes. In the next points these tools are briefly explained.

5.6.4.1 Video transcoding process:

As it has been explained, our system embeds marks into MPEG-2 video files. But, during the whole process, some video recoding processing must be performed (from AVI

to MPEG-2, FLV to MPEG-2). All these operations are done by Mplayer and FFMpeg tools [97].

5.6.4.2 Video Uploading process:

During our simulations, massive video uploading to the YouTube service has been necessary. To automate this process, some Application Programming Interfaces from Google have been used. More specifically Google published the YouTube APIs and Tools to enable the integration of YouTube's video content and functionality into websites, software applications, or devices. These APIs are available at [46].

5.6.4.3 Video downloading process:

In the same way, massive video downloading from the YouTube service has been necessary. It is true that, by means of the YouTube APIs, this process can be done but, in our implementation, the *youtube-dl* [45] Python script is used.

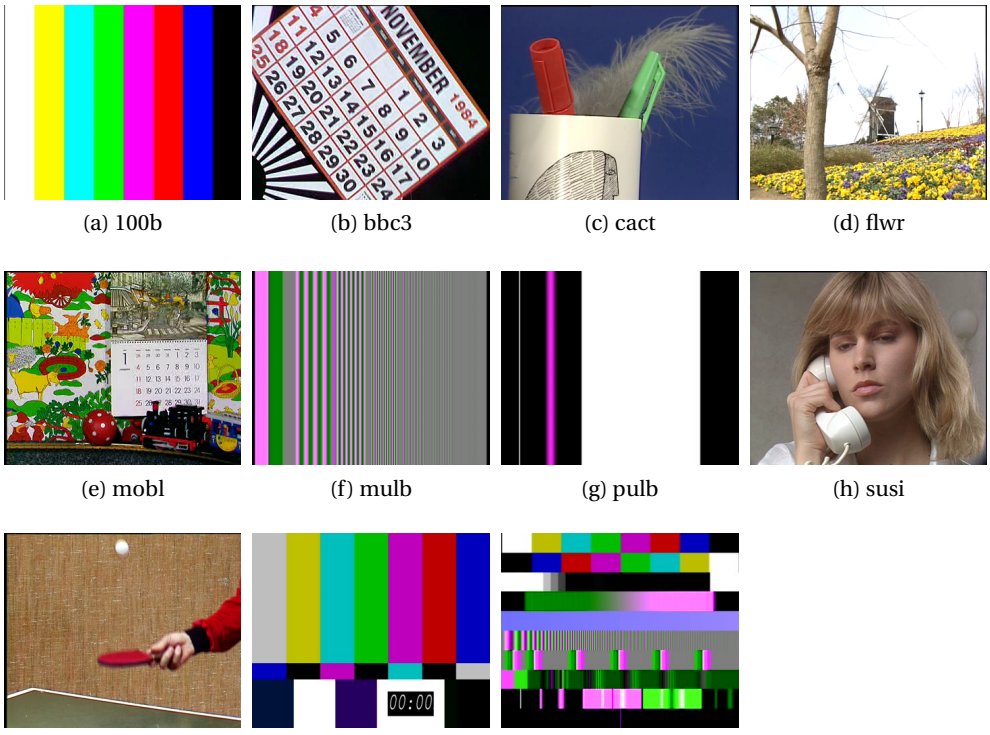
5.7 Results

In this section, 2 different kinds of results are presented. First of all, in subsection 5.7.1, some results in order to characterize the YouTube channel are presented. In this way, the relation between how the increase in watermarking power, which is the value of α, affects the bit error ratio (BER) and the PSNR. Taking into account these results, a trade-off between distortion (PSNR) and robustness (BER) appears, and the α value must be fixed in order to achieve it.

On the other hand, in subsection 5.7.2, the results from uploading videos with embedded fingerprints are shown. In this way, our work has been focused in two aspects. On one hand, in tracing the traitors in a scenario in which no collusion has been performed. The other is centered in a scenario with collusion of two users. It is clear to see that more robustness is needed in the second case due to a higher distortion. However, really satisfactory results have been achieved in both cases.

One aspect to take into account is the characteristics of the sequences used in our simulations. The corpus for Tektronix[108] has been used in order to employ very standard sequences. These originals sequences have a resolution of 352×288 pixels, a bit rate of 1.5 Mbps, 25 frames per second, and are encoded as a MPEG-2 program stream. The thumbnails of each one of these sequences are shown in figure 5.6.

Another aspect are the parameters of our fingerprinting code. As it is said before, the minimum number of users in our system must be 2^{10}. The fingerprinting algorithm can be constructed by means of fixing the parameters n and r. So we need that $r\lceil n/4 \rceil \geq 10$ and, for instance, if $n = 7$ and $r = 6$ the inequality is accomplished



(a) 100b (b) bbc3 (c) cact (d) flwr

(e) mobl (f) mulb (g) pulb (h) susi

(i) tens (j) time (k) v700

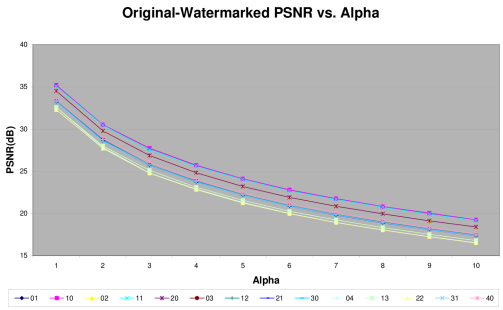
Figure 5.6: Corpus of videos used in our simulations. These images have Copyright (1996) David Sarnoff Research Center, Inc. and are availabel at [108]. All of them are MPEG-2 elementary streams with a resolution of 352×288 pixels and a bit rate of 1.5 Mbps.

$(6\lceil 7/4\rceil = 6\times 2 = 12 \geq 10)$. Finally, in our systems, a $[441, 12]$ fingerprinting code has been used (that is $[7(2^6-1), 6\lceil 7/4\rceil])$.

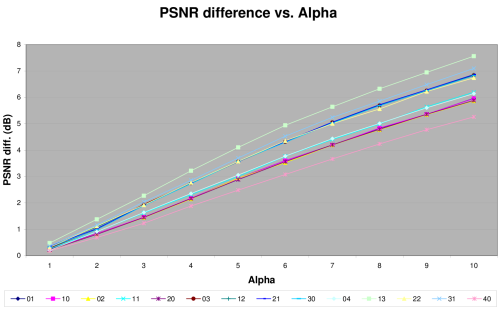
Last aspect that must be considered is that, in some cases, there are videos in which the distortion added by YouTube makes impossible to watermark in the correct way. In order to limit this effect, firstly, the PSNR between the original sequence (without any mark) and the result of sending to YouTube and retrieving it. Table 5.1 shows these values for the corpus used. In our case, sequences "mobl" with PSNR of 20,99dB and "bbc3" with PSNR of 12,44 cannot be watermarked in a correct way because they are too much distorted by YouTube.

5.7.1 How to choose the correct α.

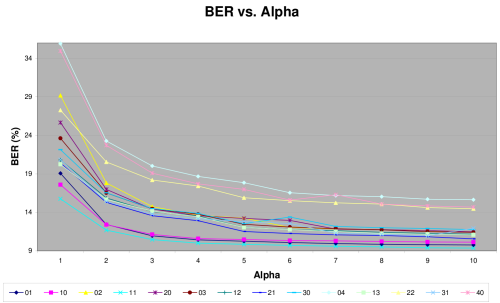
As it has been said before, there exist a trade-off between the value of α, the BER and the distortion of the resulting image. First of all, our simulations try to establish a relation between the BER and α. It is clear to see that the greater the value of α is, the greater the distortion will be. However, not all coefficients cause the same distortion



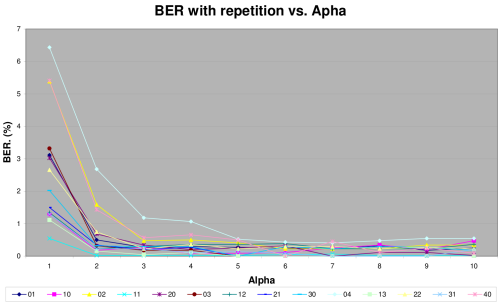
(a) PSNR between the original image and the watermarked image versus α values.



(b) Difference between PSNRs of the original image after YouTube process and watermarked image after YouTube process versus α values.



(c) Bit Error Ratio versus α values.



(d) Bit Error Ratio taking into account that the mark is embedded n times versus α values.

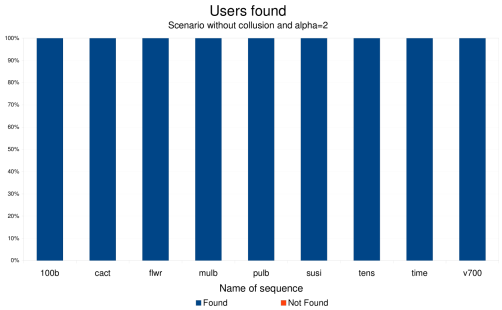
Figure 5.7: Watermarking layer performance.

Table 5.1: PSNR between original sequences and sequences after YouTube process.

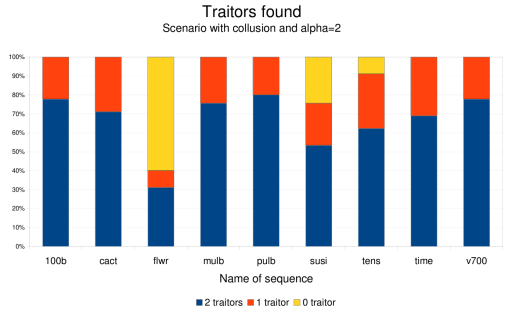
Sequence	PSNR	Sequence	PSNR
100b	43,00	pulb	37,21
bbc3	12,44	susi	30,28
cact	27,08	tens	24,62
flwr	26,11	time	34,36
mobl	20,99	v700	32,53
mulb	27,27		

by the same α value. Figure 5.7a shows, in average for each coefficient, how PSNR between the original sequence and the watermarked sequence decrease as α value increase. Take into account that only the 14 low-frequency coefficients are considered. The high-frequency coefficients cannot be used for embedding marks because it will not be robust, that is, a simple compression process will erase its values. On the other hand, the DC coefficient cannot be used because a little modification on this position will cause a really important distortion to the image.

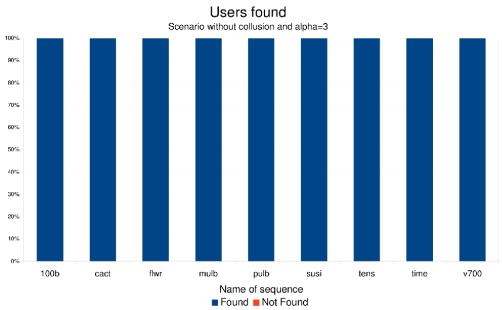
The other side of the trade-off is the BER. Similar to what happens with the PSNR,



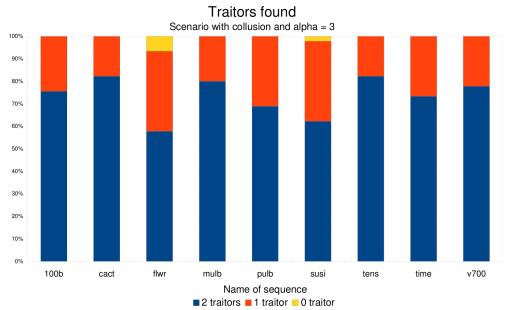
(a) Traitor tracing in a scenario without collusion and $\alpha = 2$.



(b) Traitor tracing in a scenario with collusion and $\alpha = 2$.



(c) Traitor tracing in a scenario without collusion and $\alpha = 3$.



(d) Traitor tracing in a scenario with collusion and $\alpha = 3$.

Figure 5.8: Results of traitor tracing in scenarios with and without collusion and different α values.

it is clear to see that the greater the value of α is, the lower the BER will be after the YouTube compression process. In the same way, not all coefficients are equally sensitive to this compression process. Figure 5.7c shows that lower-frequency coefficients are less affected by the compression, from BER point of view. Another aspect is that our sequences have near 51000 DCT matrixes and our fingerprints have 441 bits. In other words, if only one bit is embedded in each DCT matrix, our fingerprint can be replicated $\frac{51000}{441} \approx 115$ times. According this, figure 5.7d shows our real BER.

Finally, we need to consider the overall distortion produced by the system. A good way to measure this is by the difference of the PSNR between original sequence before and after YouTube process in one hand and, on the other hand, the PSNR between watermarked sequence before and after YouTube process. Figure 5.7b shows this difference. Note that the lower PSNR difference value is, more similar are the original sequence and the watermarked sequence.

5.7.2 Traitors retrieval performance after collusion attacks.

In this section, the traitor tracing is analyzed. The presented results have been obtained embedding the mark in the coefficient 11 with 2 different values of α (2 and

3). When no collusion is present, the user who has uploaded the video to YouTube is always correctly identified. This is shown in figure 5.8a where $\alpha = 2$ and in figure 5.8c where $\alpha = 3$.

On the other hand, if a collusion attack is performed by 2 traitors, the effectiveness of the system increases significantly as higher value of α is. In this way, if $\alpha = 3$ at least 1 traitor is identified with a probability of 95% in the worst case. However, in all cases the probability of finding the 2 traitors is higher than 50 %. This is shown in figure 5.8b where $\alpha = 2$ and in figure 5.8d where $\alpha = 3$.

5.8 Conclusions

The work presented tries to be a proof of concept that tracing traitors over YouTube video service is possible. First of all, the relation between bit error probability, watermarking robustness and distortion is deeply studied. Our study shows that a nice trade-off between these parameters can be achieved.

Next, our conclusions are applied to the problem of traitor tracing. In this way, we use the watermarking layer (configured taking into account our results) with a fingerprinting code. It is shown that this fingerprinting code can trace traitors if no collusion is performed, with a really low distortion. If collusion appears, the traitors could be also traced with a low distortion. Besides, the proposed system does not allow false positives by design, that is, innocent users cannot be framed.

CHAPTER 6

DEVELOPMENT OF A PLATFORM FOR THE COPYRIGHT PROTECTION OF MULTIMEDIA CONTENT

This chapter presents a software platform that consists of a combination of watermarking and fingerprinting techniques to help protect authorship and copyright of multimedia content. The software that has been developed, provides content distributors and authors with a trusted system that allows the former to develop new business models while preserving the authorship rights of the latter. The proposed system offers the distribution of multimedia content via a Web platform, providing mechanisms for tracing dishonest users that illegally redistribute their content.

6.1 Introduction

Distribution and playback of digital content (image, audio, video, text, ...) using a personal computer has become a trivial matter. This ease of use leads to problems concerning the protection of copyright and distribution rights. The scientific community has been devoting extraordinary efforts to devise a copyright protection system that helps to prevent these illegal practices. Of course, data encryption provides protection up to the transport layer, but when an authorized dishonest user decrypts his/her content, nothing can prevent him/her from distributing the content without any concern to be indicted or punished for his/her actions. Watermarking schemes appear to be a possible solution to this problem. These schemes allow to embed information about the owner (brand) in a given original content. Unfortunately, watermarking alone does

not protect against illegal re-distribution. However, when combined with digital fingerprinting techniques [11], both mechanisms offer a good solution to deter potential fraudulent users from re-distributing illegal content. To make distribution systems work properly, the solution must be robust to attacks such as image processing operations, lossy compression, geometric transformation, combination with additive noise, and/or collusion attacks.

The work proposed in this chapter consists on the design and implementation of an empiric and portable platform that satisfies the following objectives:

- Provide a public and secure platform capable of tracing users that perform illegal digital video re-distribution.
- Check on a practical level the robustness of watermarking and fingerprinting algorithms working together.

The platform is presented as a content manager system (CMS), liaising between authors and customers. The aim is to provide a trusted environment that offers protection against illegal re-distribution of authorized content. The protection provided is achieved by means of combining watermarking and fingerprinting techniques for the case of MPEG-2 digital content. The generated watermarks, as well as the insertion algorithm have been designed in such a way that guarantees robustness against common attacks, while avoiding content degradation.

The chapter is structured as follows; Section 6.2 describes the working scenario, paying special attention to the platform functional and architectural key points, such as security, modularity and flexibility. Section 6.3 discusses watermarking mechanisms, as well as the implementation internals of the watermarking layer. Section 6.4 describes the different entities and their interrelationship. Finally, conclusions are given in Section 6.5

6.2 Working Scenario

Consider an scenario where a digital content provider offers its users copies of digital video for their amusement. Of course, the provider desires as much protection as possible against illegal re-distribution of these files to third parties. Authorized users access the provider's Web portal to download their favorite movies. Once the user has purchased a given content he is free to illegally distribute it via P2P platforms or upload it to content distribution platforms such as Youtube or Fileserve, among others. The content provider, after detecting that one of its products is freely available on the Internet realizes of a security leak in his business chain, but there is not much more that he can do. It would be nice for him/her to at least identify the user that purchased the

copy that has been illegally re-distributed. This user may have incurred in a contract fault by distributing content without specific authorization, allowing the company to take internal actions over the user's account, regardless of the legal actions that could be taken over the physical person.

The work we present aims at filling the gap between detection of a dishonest distribution and the actions taken on the accused individual. This may be performed by identifying the owner of the illegally redistributed copy.

So, in order to be able to identify the owner of a given a video file, there must be an association between the user and the copy. This can be achieved by means of inserting some information inside the content that may identify the user once the content is recovered, this embedded information is called a mark. Of course there are some requirements [27] that this mark has to fulfill:

- **Perceptually invisible**: perceptual invisibility is a wanted characteristic of the mark. The process of embedding the information into the data is really important. In one hand, the introduced mark cannot be perceptually noticed by viewers. On the other hand, once introduced it cannot be easily removed, at least without much degradation of the original data.

- **Statistically invisible**: each mark must be statistically uncorrelated to any other mark or the content itself.

- **Robustness**: The mark must be robust to modifications as, for example, image processing.

- **Unambiguous**: A mark should unambiguously identify the authorized owner of the content in which has been introduced.

- **Complexity**: The complexity of a mark is proportional to its degree of efficiency. A more complex mark is also more difficult to remove.

- **Low error bit rate**: When recovering the mark it has to be guaranteed that it is "impossible" to obtain a mark that belongs to a user different than the original one. "Impossible" here means with very low probability.

Clearly there is a need to embed customer information into delivered multimedia products to ensure copyright protection. The presented platform pays special attention to this concern and proposes the usage of user and server Public-Key certificates to protect user-to-platform and platform-to-platform transactions.

6.3 Implementation Details

The aim of this platform is to guarantee the correct management of digital rights (both copyright and distribution rights). This goal is accomplished by means of watermarking algorithms and fingerprinting codes. In this section, the implementation of these mechanisms is discussed. On one hand, the watermarking layer discusses how the marks are embedded into the video content. On the other hand, fingerprinting algorithms generate the embedded marks.

6.3.1 Watermarking Layer

Watermarking is a technique for embedding information into data files. When using watermarking schemes, all the copies of a file contain a mark, that can be recovered at any time.

From the detection point of view watermarking systems can be classified in two groups: blind watermarking and non-blind watermarking. The first one assumes that the original content is not needed to perform the recovery of the mark. The second algorithm assumes that the original content can be used by the decoder in order to improve the performance of the whole system. Since in our working scenario, it can be assumed that the decoder has access to the original content, we have decided to use non-blind watermarking. An extensive analysis about the drawbacks of blind watermarking systems is given in [28].

Usually watermarking is performed in the frequency domain. There are different domain transforms we can use, for example, the Fast Fourier Transform (FFT) or the Discrete Cosine Transform (DCT). In [28] has been shown that DCT gives a better answer to image watermarking needs. After the transformation into the frequency domain a mark is added to the signal. The inverse process transforms again the signal to the spatial domain.

There are many advantages of placing the watermark in the frequency domain. As it also occurs in the spatial domain, different frequency values can be modified. Each frequency coefficient has a perceptual capacity. This perceptual capacity represents the capacity of each coefficient to receive additional information (watermark), without introducing perceptual differences in the content. Note that the Human Visual System (HVS) model naturally works with frequencies. So working in the frequency domain helps adjusting the addition of information together with the HVS sensitivity.

6.3.2 Fingerprinting Layer

Roughly speaking, the fingerprinting layer will generate a different mark for each distributed copy of a multimedia content. This mark will be embedded by the watermark-

ing layer into a verbatim copy of the original content in order to generate a new copy for a particular user. When one of these copies is illegally re-distributed, this mark is extracted, and by means of a tracing algorithm, the traitor (the fraudulent user) is identified.

This first approximation is more or less achievable but the real fingerprinting problem consists in finding, for each copy of the original content, the right set of marks that prevent collusion attacks. A collusion attack is performed by a set of users that have acquired different copies of the same multimedia content. Note that, in order to identify every user, every single mark embedded into each copy has to be different. Since the attackers decide to cooperate, they have access to different copies of the same content so they can generate a pirate copy of this content by mixing their copies. In this scenario, the pirate copy will have parts of each original mark but it will be different from any of them. So the aim of a fingerprinting code is find the set of attackers that have taken part in a collusion attack.

Codes that are robust to collusion attacks are called collusion secure codes. The construction of collusion secure codes was first addressed in [11]. In that paper, Boneh and Shaw obtain ($c > 1$)-secure codes, which are capable of identifying a guilty user in a coalition of at most c users with a probability ϵ of failing to do so. The algorithm composes an inner binary code with an outer random code. Therefore, the identification algorithm involves the decoding of a random code, that is known to be a NP-hard problem [6]. Moreover, the length of the code is considerably large for small error probabilities and a large number of users.

To reduce decoding complexity, Barg, Blakley and Kabatiansky in [6] used algebraic-geometric codes together with separating codes to construct fingerprinting schemes. In this way, their system reduces the decoding complexity to $O(poly(n))$ for a code with length n. In [38], Fernandez and Soriano constructed a 2-secure fingerprinting code by concatenating an inner $(2,2)$-separating codes with an outer IPP code (a code with the Identifiable Parent Property), and also with decoding complexity $O(poly(n))$. In [107], Tardos presents a code which has a length of $O(c^2 log(n/\epsilon))$ and is ϵ-secure against c pirates over n users. These codes represented an important improvement compared to the Boneh and Shaw proposal.

In our implementation, the code and algorithms presented in [38] have been used as fingerprinting codes mainly because in our scenario there is a small number of users and the size of the attacking coalitions is assumed to be small also.

6.3.3 Implementation Details of Digital Rights Protection

The watermarking process requires a considerable amount of CPU running time and computer memory, so it is important to use an efficient encoding/decoding process

to add the mark to the video. This was the most important factor when deciding how to implement a functional marking software. Implementing an MPEG2 encoder/decoder is difficult and would require a lot of knowledge to make it efficient. To solve this problem we decided to implement the marking algorithm on an already working MPEG2 encoder/decoder, FFmpeg.

FFmpeg was chosen because it is licensed under the GPL, and because it is highly efficient. The work presented in this chapter will be mainly focused on adding a new set of filters that allow the embedding and recovery of a mark inside a video stream.

Libavfilter is an FFmpeg library, designed to simplify the creation of filters that can be applied to video processing, regardless of the format of the input and output videos. This is accomplished by decoding the input stream and passing it to the filter a single frame at a time, then the processed frame is coded and saved in the output stream, this flow is represented in Figure 6.1.

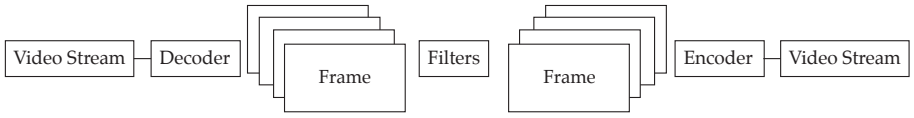


Figure 6.1: Libavfilter working flow

This flow allows developers of filters to focus on the processing of the image, without handling any of the decoding, encoding or muxing. This is specially interesting when the filter has to be applied in the spatial domain, but in our case the filter has to be applied in the DCT domain. This might seem like a problem, since AVFilter provides us with an image divided in 4 planes. The more usual procedure for modifying the frame would be to perform the DCT of the pixels, modify the desired coefficients and perform the IDCT to convert the image back to the spatial domain. This however can be simplified with the following DCT property.

Definition 6.1. ***(from [95])*** *Given the integrable functions $f(x)$, $g(x)$ and $h(x)$ we denote their discrete cosine transforms by $\hat{f}(\xi)$, $\hat{g}(\xi)$ and $\hat{h}(\xi)$ respectively:*

For any complex numbers a and b, if $h(x) = a \cdot f(x) + b \cdot g(x)$, then $\hat{h}(\xi) = a \cdot \hat{f}(\xi) + b \cdot \hat{g}(\xi)$.

Based on Definition 6.1, the embedding process is less costly, since we no longer need to perform the DCT of the original image, instead we can create and empty matrix, fill the desired positions with the marking coefficients, perform the IDCT of the matrix and add it to the original image as seen in (6.1).

$$\underbrace{\begin{bmatrix} 20 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ -30 & \cdots & 40 \end{bmatrix}}_{Marked\ DCT\ positions} \xrightarrow{IDCT} \underbrace{\begin{bmatrix} 5 & \cdots & 2 \\ \vdots & \ddots & \vdots \\ -3 & \cdots & 9 \end{bmatrix}}_{IDCT\ of\ marked\ positions} \xrightarrow{Add\ to\ original} \underbrace{\begin{bmatrix} 200 & \cdots & 154 \\ \vdots & \ddots & \vdots \\ 8 & \cdots & 76 \end{bmatrix}}_{Image\ after\ watermark} \tag{6.1}$$

Once the watermarking and recovery filters were implemented, some tests were conducted in order to prove the robustness of the system. A sample movie with the following properties was used to conduct these tests:

Movie sample	
Duration	101s
Size	47,6MB
Bitrate	4000kb/s
Framerate	23.98fps
Dimensions	848x480
GOP Size [1]	12 frames

This video sample corresponds to a movie trailer freely available on the Internet. All tests where performed trying to simulate an scenario as close to real life as possible. Therefore, the sample was re-encoded several times. For instance, in the scale tests, the movie was first watermarked, then scaled to the desired size. Then it was scaled back to the original size and finally it was processed in order to recover the embedded mark (the movie was decoded and encoded 3 times). The tests were performed this way to simulate the behavior of a dishonest user, who might try to scale the movie in order to remove the embedded mark. Once the movie is recovered, it has to be scaled back and then compared with the original one in order to extract the mark.

The table with the result of the simulations contains the Bit Error Ratio (BER) of each recovered watermark. This rate is the number of erroneous bits divided by the total number of bits that a mark contains.

As we can see in the results table, we obtain a very low BER on most tests, and the only way to introduce a significant degradation to the mark also has a very high impact on the image quality, rendering the sample unwatchable (a blur filter with a factor higher than 7 or a Gaussian filter with a factor higher than 12). Hence we can assert that the implementation of the watermarking algorithm is robust, portable and efficient.

[1] Distance between consecutive I-Frames

Table 6.1: Tests results for movie sample

	Mark strength ^{2}								
	40	35	30	25	20	15	10	5	1
Mark and recover (no filter)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.01
Gaussian filter factor 3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2
Gaussian filter factor 5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.32
Gaussian filter factor 7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.4
Gaussian filter factor 9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.01	0.41
Gaussian filter factor 11	0.0	0.0	0.0	0.0	0.0	0.01	0.02	0.1	0.44
Gaussian filter factor 13	0.0	0.0	0.0	0.0	0.004	0.02	0.12	0.23	0.48
Gaussian filter factor 15	0.0	0.006	0.01	0.02	0.07	0.11	0.17	0.3	0.53
Blur filter factor 3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3
Blur filter factor 5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.07	0.45
Blur filter factor 7	0.28	0.23	0.26	0.18	0.16	0.24	0.37	0.42	0.49
Scale to 706x400 (5/6)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.21
Scale to 636x360 (3/4)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.32
Scale to 424x240 (1/2)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.37
Scale to 282x160 (1/3)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.01	0.51

6.4 Entities and Collaboration

In order to achieve the enumerated requirements, the proposed platform has been divided into the following entities:

- **User**: consists on the physical person that interacts with the system via the Web interface. Later, the different roles a user can take will be introduced.
- **File server (FS)**: stores the original files as well as the watermarked copies.
- **Marker server (MS)**: provides marked copies of original files.
- **Tracker server (TS)**: manages the business work flow.
- **Web server (WS)**: provides a Web interface to all platform functionalities.
- **Certification authority (CA)**: manages the certificates that will be used in the communication process, and has real-time certificate revocation status information.

The interaction between these entities provide the end-users with a protected copy of a digital video file. The flow of the platform is the following: *verification, protection* and *delivery*.

During the *verification* phase, both client and platform, must validate each other's credentials in order to allow further transactions to take place.

Considering the nominal usage of the copyright protection platform, someone aiming to request digital contents, the user, is required to own a private certificate

(X.509) delivered by the internal certification authority (CA). This authority provides valid certificates to both users and services. This way, all communications carried out with and inside the platform are secured using the provided certificates.

When contacting the platform public URL through a common Web browser, the client will be requested to specify the certificate to be used in the negotiation. Meanwhile, the platform will provide its server certificate for the client to verify. This mechanism allows the mutual authentication that is going to be carried out in the negotiation phase.

Once the negotiation phase has been finished successfully, the server presents customized contents to the user. As the user presents a valid certificate, the server identifies the user roles and according to this, prepares the layout. Not registered users are considered new clients and are registered as such during the first negotiation phase.

Once this phase is concluded, the user may request a digital copy of an offered product. At this stage the *protection* phase takes place. During this phase, the system must provide a protected copy of the requested product to the client. So, the WS receives the request to download a copy of a product identified by a hash. The platform makes use of hash identifiers in order to avoid inference of other product identifiers. At this time, the WS checks internally if the user already purchased a copy of the requested film, if so, the WS contacts the FS to get the already marked copy of the product. This means that a user that owns a unique certificate, can only have one marked copy of each original.

If the WS does not find a copy already purchased by the requester, it will contact the TS to request a new copy. The TS generates then a new copy by first, generating a new mark and secondly by asking the MS to embed the given watermark in the original product, generating a unique copy that the TS will then assign to the requester. At this time the WS finds as a result a marked copy assigned to the user.

The *delivery* phase starts when the TS notifies the WS that a new copy has been assigned to the requester, passing the copy hash to the WS as well as the location of the FS where the copy is located. The WS contacts the FS to download the copy and forwards it back to the user.

Note that, any interaction between services is carried out using the service certificates hosted on each service instance. Hence, the WS contacts the TS, and both perform a mutual authentication before the transaction is carried out. Every time a connection is established, both ends verify each other's certificate against the CA OSCP service [85].

6.4.1 Platform Functionalities

What has been presented above consists on the basic flow of a product request. Moreover, the platform provides a set of functionalities for the different actors that may interact with the system in order to be able to perform the flow presented in the previous section. Indeed, the platform proposes four different roles:

- **Administrator**: manages and configures the platform through the administration functionalities.
- **Client**: requests new digital contents and navigates through already downloaded resources.
- **Guest**: navigates through published contents but cannot download them.
- **Content manager**: manages the digital contents and stocks.

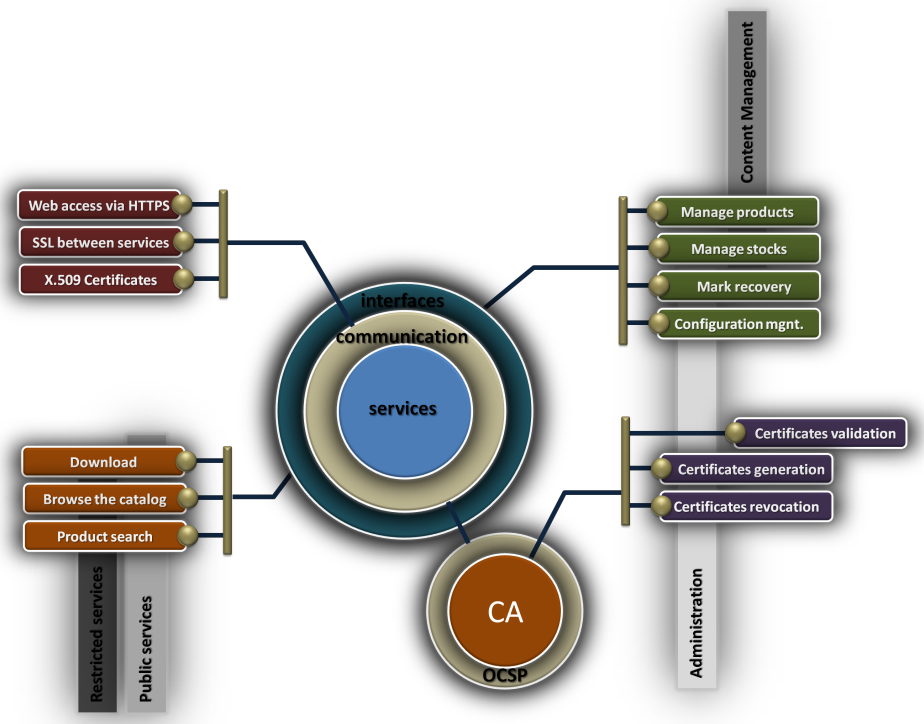


Figure 6.2: Graphical representation of platform main functionalities.

Figure 6.2 presents the functionalities available for each of the proposed roles. Hence, users with administrator's role have full access to the administration functionalities;

- **Mark recovery**: Allows the extraction of the mark embedded in a digital content. Via a web form, the user submits the digital copy with an embedded mark and specifies, from a list of products, the name of the original content that the copy was made from. As a result, the user obtains a link to the copy managed internally, together with information about the registered user to whom the copy was assigned.

- **Configuration management**: Allows the user to adequate the platform to the deployment context. There exist static and dynamic parameters that can be modified before and after the platform is started-up, therefore affecting the platform behavior.

- **Certificate generation**: As mentioned previously, any actor of the system must own a valid digital certificate generated by the certification authority (CA). By means of this functionality, the user can generate new certificates for new clients or new server instances. This functionality is offered directly by the CA through the Web interface using the proper client certificate.

- **Certificate revocation**: In some circumstances, under a security compromise, it may be recommended to revoke certain certificates. The CA offers a Web interface for that purpose. Considering that all platform services verify client certificates on each request, a certificate revocation has an immediate effect, offering fast reactivity to potential security vulnerabilities.

On the other hand, users with content management's privileges have access to the following functionalities;

- **Manage products**: The platform offers a complete Web interface to manage multimedia contents. A user with proper privileges is able to navigate through all the products in the catalog browser, search for specific products via the search form, modify certain products' attributes and upload new contents.

- **Manage stocks**: Although stock management will be introduced later, a user with content manager privileges has access to the Web interface to check the products stock level. By knowing the actual stock level for each product, and analyzing the latest client interests, the content manager can generate new copies of certain products by modifying its minimum stock level.

The last set of functionalities are available to those clients with (Restricted services) or without (Public services) a valid user certificate.

- **Browse catalog**: Any user consulting the web site, having or not a user certificate, can visualize the catalog browser that presents available categories. Categories are defined and maintained by the content manager and serve to group

the movies according to its typology. For instance, the platform defines twelve categories. Among others we have: horror, adventure, animation, drama, comedy, etc. The catalog, located on the right column, shows all categories with the number of products available. By following one of the categories link, all products belonging to it are presented.

- **Search products**: The platform provides all users, having or not having a user certificate, with a fast search form. Through this fast form, the user may enter some key text related to the product he/she is looking for. The products matching the search criteria are presented.
- **Download products**: The download functionality is only available to registered users, that is, those having a valid user certificate. So, after a user has found the desired product, either via the search form or the catalog browser, a request can be made via the download link. A copy of the original product, with an embedded mark will be returned. Successive downloads of the same product return the same copy, stored in the platform repository.

6.4.2 Stock Management

Considering that the main functionality of the copyright protection platform is to provide marked digital copies, the management of the product life cycle is a very important aspect to consider. The content manager is responsible for the product life cycle management, providing the original copies of the video files as well as the support attributes that serve to define the product: name, description, video cover, categories and minimum stock level. As already mentioned earlier, when a user requests a new product from the catalog, a watermarked copy is provided. But the watermarking process is not immediate, therefore, if the system generates copies of the original product on-the-fly, the user would be waiting for too long. As an order of magnitude, watermarking a DVD quality sample with a bit rate of 7500 kb/s and a duration of 110 minutes (video stream size of 4,89GB) on an Intel(R) Core(TM) Xeon E5645 CPU @ 2.40GHz takes 17 minutes using a single thread and 10 minutes when using 4 threads, long enough to avoid having the user waiting for his/her brand new purchase.

To avoid the generation of copies based on user requests, the platform generates a given number of copies when the content manager uploads a new product. So, according to the “minimum stock level” attribute of the new product, the system generates this precise number of marks and embeds them into the same number of copies.

Later, when a user requests a copy of a given content, the platform identifies a non assigned copy. Such copy is then assigned to the requester and returned as an attachment. Afterwards, a stock level verification is performed on all existing products. The stock level of a particular product consists on the number of generated copies not yet

assigned to any user. When the stock level of a product reaches the minimum defined as a parameter on a product upload, new copies are created as to reach such level. On the other hand, the content manager has the ability to control the stock level manually. Therefore, the minimum stock level of certain products can be modified to satisfy a specific user demand.

Although the system architecture will be presented in following section, the marking service consists on a number of modules that can provide copies of original products concurrently. More over, according to this design principle, if multiple users request the same product, and possibly deplete the stock, the system will generate new copies in order to regenerate the stock level by launching multiple requests to available watermarking services, and thus satisfying peak demand points.

6.4.3 System Architecture

The proposed platform has been designed and developed as a distributed JEE application. The platform is basically made up of two software modules; web modules and service modules. Figure 6.3 shows the basic components that conform each module.

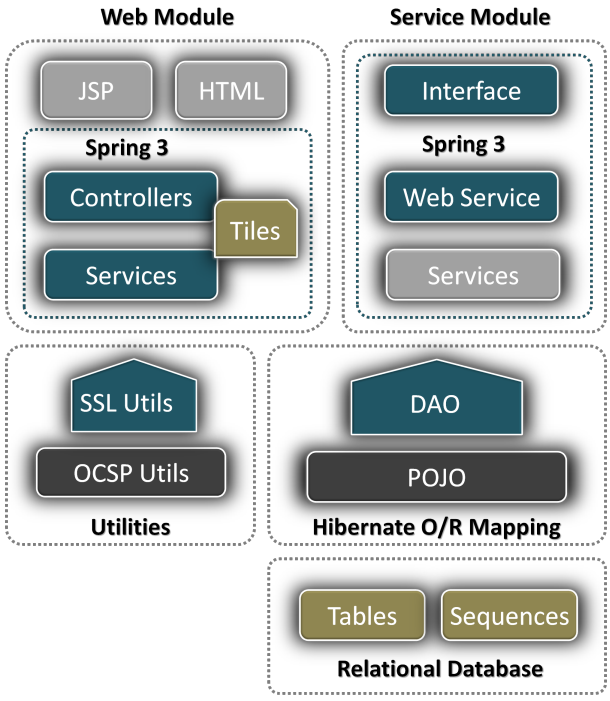


Figure 6.3: Diagram of the generic platform modules

Both modules are build up around the Spring 3 JEE framework [3] which powers up the development cycle of Web applications. Moreover, the database repository is ac-

[3] http://www.springsource.org/

cessed via Hibernate [4] components which are published as a JAR library. The Object/Relational (O/R) mapping components consist on the basic Plain Old Java Objects (POJO) that map the relational database tables to Java objects using Hibernate Java Persistence Annotations (JPA). The upper layer offers a set of Java objects to facilitate the POJO management, the so called Data Access Objects (DAO).

The Web module consist on a set of controller components in charge of managing the user requests, each controller maps a set of URLs the user may follow when accessing the Web portal. The controller dispatches the business logic execution to the proper services, when concluded, the services provide information in the form of POJO that are rendered dynamically by Java Server Pages (JSP). The whole view layer is build up using Spring 3 tiles, which facilitates the graphical design and maintenance of the Web portal.

Although the service module architecture is very similar to the web module's, the interface changes from HTTP/HTML to HTTP/SOAP [13]. This way, the platform services publish their functions externally as standard Web Services. All platform components are packaged as Web archives (WAR) and can be deployed on any servlet container.

The available type of services were slightly introduced on the preliminary sections, indeed, the platform offers, besides the TS, a File Server (FS) and a Marker Server (MS). Although there is a single instance of the TS, there may be multiple instances of FS and MS services depending on the system needs. The platform proposes a service registration process in order for a process to be active and public.

So, when a service starts-up, the first thing it does is to contact the TS to register on duty. When the connection is established between the new candidate and the TS, and by means of mutual authentication, the TS validates the candidate certificate against the CA, as explained in the previous section. If the validation is successful, the TS registers the service in a UDDI [7] (Universal Description Discovery and Integration) repository. This will allow the TS to query for active services later. When a service closes in a controlled manner, the de-registration process takes over, eliminating the service from the UDDI repository.

Figure 6.4 shows the deployment diagram of all components considered in the platform. Each dashed box represents a deployment container, indeed there are two different containers; Apache Tomcat 7 [5] and JBOSS 5 [6]. Tomcat holds all platform services while JBOSS holds the Certification Authority (CA) which is implemented using an Open Source PKI Certificate Authority called EJBCA [7].

[4] http://www.hibernate.org/
[5] http://tomcat.apache.org
[6] http://www.jboss.org/jbossas
[7] http://www.ejbca.org

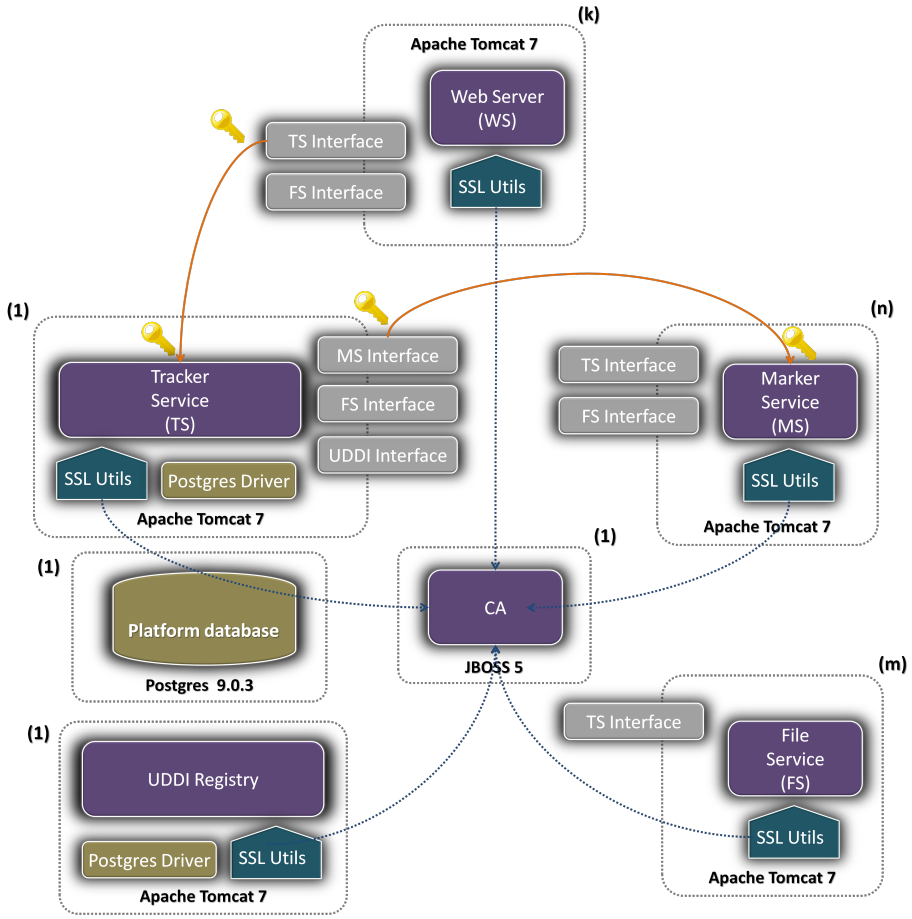


Figure 6.4: Deployment diagram of the services proposed in the platform

The figure shows as well the communication protocols used between the components. Basically, the figure shows only two communication protocols; OCSP and SOAP. OCSP, which stands for Online Certificate Status Protocol, and it is represented as a blue dashed line, is carried out between all platform services and the CA to check the validity of a given certificate.

The rest of interactions between services are carried out using the SOAP protocol. For simplicity, the figure shows only two orange lines representing the SOAP communication between services. Moreover, the representation of the public interfaces in the Web Service clients helps infer the rest of missing lines. As an example, on certain user requests, the WS contacts the TS via the TS Interface, this interface consists on the Java classes generated at client side given the server's WSDL [32] (Web Service Definition Language). Therefore, any module holding a TS Interface requires an orange line between the interface and the TS. The TS delegates the task of marking to the MS, which is done via the MS Interface.

Note as well that all communication carried out under SOAP protocol requires

both client and server valid certificates. This allows for mutual authentication and data encryption in both directions.

The figure represents as well the number of permitted instances for each service, represented as a number inside brackets. As mentioned before, the number of FS (m), WS (k) and MS (n) instances can be configured depending on the context scenario, and it has to be done at deployment time.

6.4.4 Platform User Interface

The figure 6.5 shows the user interface presented to an authenticated user when first login. The left figure 6.5a presents a set of the most popular product covers. The cover image link redirects to the product detailed information 6.5b. Below the cover flow, all products are presented, by default, in name alphabetical order. The sort criteria can be modified through the combo options located on top of the products list.

(a) Most popular product covers (b) Product detailed information

Figure 6.5: System welcome page and product detailed information view

The product information page presents detailed information not available on the products list page (Figure 6.6). Depending on the user role, the page presents basic or full attributes. The basic attributes are available to clients and provide information about the product itself, such as the name of the film, a description, the categories it belongs to, the author, the publication date, the purchase date in case already has been purchased and the product size.

The administrator role can visualize extra product attributes; minimum stock level, product sequence number (considered to identify the next fingerprint to embed on the next copy), the DCT matrix positions where the watermark is inserted, distance between marked frames, coding, mark depth, and specific coefficients for watermark generation. Moreover, if the user has the content manager role, the view provides a link to modify the product attributes.

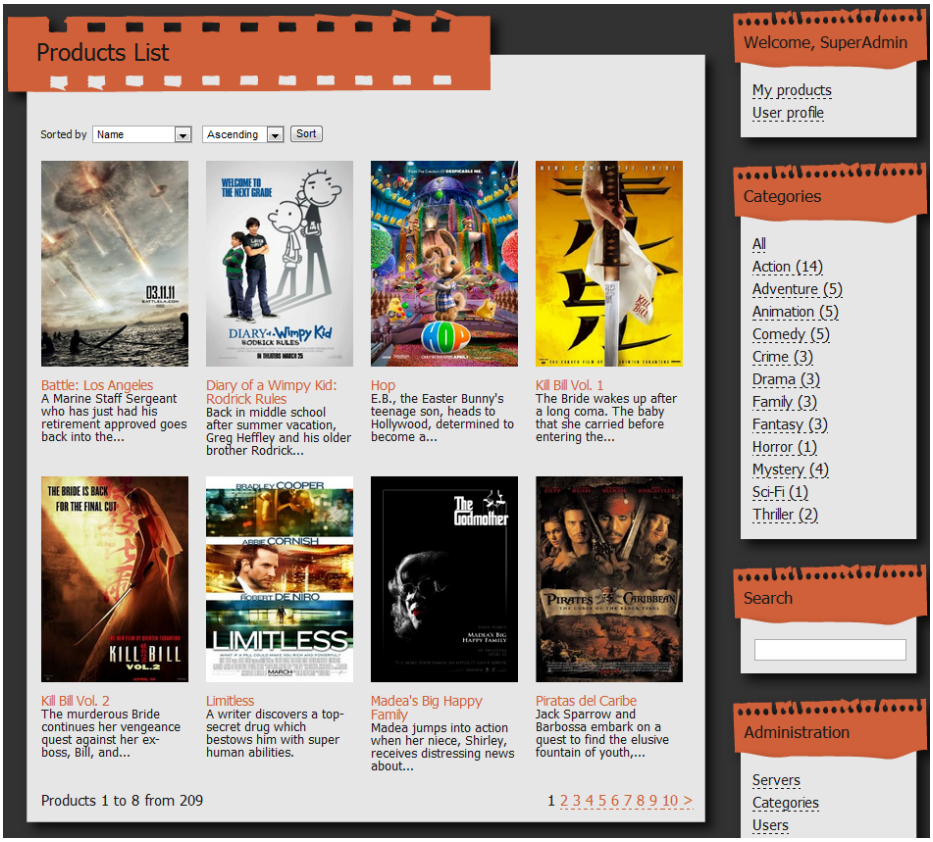


Figure 6.6: Products list and available menu options (right side)

The Web page displays the menu options grouped in four categories, as presented in Figure 6.6; user specific options, categories, search and administration. Obviously, depending on the user role, some of the options may be hidden, as are specific for administrators or content managers. For instance, a basic user or a guest has access to user specific options, categories and search options. On the other hand, administrator and content manager roles have access as well to the administration group.

The administration menu provides a set of options for the platform management. The following is only available for administrators and content managers:

- **Servers**: this option provides the administrator with a list of all available servers offering a platform service, that is, a tracker service, a marker service or a file system service. The list of services is extracted from the UDDI registry and presented as a plain list with some extra attributes. Through this list, the administrator can check the number of existing services of a given type.

- **Categories**: the content manager can update the list of categories through this functionality. Categories can be extended or modified to satisfy the available

offer. Categories are offered to the content manager to classify new products or to update existing ones.

- **Users**: the administrator can, through this view, take a look at the list of registered users. This list presents the user certificate name, the serial number and the state (whether the user is active or not). Through this list, the administrator may cancel a user account by de-activating it, this will trigger a certificate revocation as well as an account cancelation. The following user login attempts with a deactivated certificate will be denied.

- **Parameters**: Some of the services activity can be parameterized, therefore adapting the system to certain context conditions. These parameters can be managed through this view. There are parameters related to the watermarking process as well as the Web interface layout.

- **Stock**: as already explained in Section 6.4.2, the platform provides a functionality for the manual management of stocks. Through this link, the content manager visualizes the list of products with the number of available watermarked copies as well as the minimum configured stock level. At any time, the minimum stock level can be updated to force a later stock regulation.

- **Upload**: the content manager has the responsibility of providing new content by uploading, describing and categorizing multimedia contents. This link offers a Web form to define the characteristics of the new product that will be available to registered users for download.

- **Mark recovery**: in order to ensure copyright protection the administrator and/or content manager can use a tool to recover the watermark that has been embedded on a product copy. This link provides access to a Web form where the user introduces the recovered copy location and the name of the original product the copy was made from. The result of the analysis process consists on the list of users involved in the dishonest copy distribution. There may be more than one dishonest user if a confabulation process has taken place.

6.5 Conclusion

This chapter has presented the implementation of a platform for the delivery of copyright protected digital content. The copyright protection is based on a combination of watermarking and fingerprinting techniques that allow the generation of protected copies that can be distributed in a trusted environment. The dishonest redistribution of the purchased contents can be traced by means of watermarking extraction and

fingerprinting analysis. Both techniques result in the identification of the malicious users involved in the unauthorized redistribution process. The solution is presented as a distributed JEE application based on standardized frameworks such as Spring and Hibernate. As a result the system offers loosely coupled Web services with a high cohesion. The independence and modularity of the services invite for good maintainability and high scalability. The platform presents a demonstrable evidence of a copyright protection system that proves the feasibility of the implementation of watermarking and fingerprinting algorithms on a real life scenario.

Part IV

Contributions related to Mobile Agent Protection

CHAPTER 7

EXECUTION INTEGRITY OF MOBILE AGENTS IN INTRUSSION DETECTION SYSTEMS

In an agent's environment, the most difficult problem to solve is the attack from a platform against the agents. The use of software watermarking techniques is a possible solution to guarantee the integrity of mobile agents' execution. In this chapter these techniques are use to protect Intrusion Detection Systems (IDS) based on agents. To achieve this goal, two different approaches are proposed. On one hand, a watermark is embedded in the agent by means of the Dynamic Graph Watermarking algorithm [23, 22]. This watermarking is stored into the structure of a graph which is created during the agent execution. This graph could be retrieved throughout the agent life in order to check that it is being built correctly. When the agent's execution is modified, the graph is altered so the system can conclude that an attack has been performed. On the other hand, the use of Self-validating Branch-Based Software Watermarking algorithm [87] is proposed in order to embed a matrix of marks in each transceiver of the IDS, that are the software entity responsible of control all the agents which have been executed by a host. Every time that a CIA arrives in a host, it requests the fingerprinting mark to the transceiver and this mark is sent to the monitor who can decide if the mark is correct. If any error occurs, the platform initiates the necessary actions to isolate the compromised host. Moreover, obfuscation techniques are included to difficult a possible code analysis by an unauthorized entity.

7.1 Introduction

The security of systems based on software has become an important subject due to most of them controls critical infrastructures like disaster prevention centres, intelligent buildings, planes' functions automation, etc. So, many human lives and huge amount of money could depend on the confidentiality, integrity and availability of these critical systems. There are several tools to achieve these security requirements such as firewalls, honeynets, honeypots, intrusion detection systems, etc. Due to the high dependability of the systems in this type of tools, they become objectives susceptible of being attacked and therefore in critical systems that also need to be protected.

Particularly, the Intrusion Detection Systems (IDS) have as a goal to detect suspicious activities and prevent a network or system from possible intrusions at the moment when they happen. Therefore, it is important to keep in mind the integrity of the information, authentication and access control. The different entities that compose the IDS need to be communicated among them and cooperate to achieve the system's goal. So, the use of agents inside IDS has been proposed due to they can perform simple actions, that joining them resolve complex tasks [92].

On the other hand, one of the reasons that have held back the generalized use of the mobile agents is precisely their security. In this chapter we focus our attention in the agent's protection as part of an IDS. In particular, we improve the work presented in [94] in order to make the system more resistant against replay attacks.

7.2 Background

In an IDS based on autonomous agents it is necessary to combine different tools to guarantee the required security level. We propose to use software watermarking and software obfuscation techniques. Likewise, we have analyzed possible threats to provide a solution.

7.2.1 Software watermarking and fingerprinting

Watermarking techniques have been basically used in the protection of digital contents. With these techniques, some information (usually called mark), is embedded into a digital content like video, audio, software, etc...The main objective is to keep this information imperceptible in all copies of the content that is protected in such a way that the author can later demand the authorship rights over these copies.

In software watermarking, the mark must not interfere with the software functionalities. The mark can be static, when it is introduced in the source code, or dynamic, when it is stored in the program execution states. In the same way, the aim of software fingerprinting techniques is to identify the author of copies, as in watermarking

scenarios, and also identify the original buyer of each copy. In other words, a different mark is embedded in every copy before distribution. The main attack to fingerprinting schemes is the collusion attack, meaning that, some malicious users compare their copies and they can try to construct a new copy with a corrupted mark which can not blame any of them.

In the scenario presented in this chapter, the fingerprinting techniques are used to include different marks in each copy. As a consequence of using fingerprinting and watermarking techniques, these marks will be imperceptible against inspection attacks and it provides a consistent tamperproof protection.

7.2.2 Intrusion Detection Systems

An Intrusion Detection System (IDS) tries to detect and alert about suspicious activities and possible intrusions in a system or particular network. An intrusion is an unauthorized or undesired activity that attacks confidentiality, integrity and/or availability of the information or resources. In order to reach its goal, IDS monitor the traffic in the network or gets information from another source such as log files. The IDS analyses this information and sends an alarm to the system administrator. The system administrator decides to avoid, correct or prevent the intrusion.

The basic architecture of IDS is conformed by the data collection module, detection module and response module [47]. The data collection module contains the event generator sub-module which can be the operating system, the network or a particular application. The events generator sends the packets to the events collection submodule which collects the data and sends the information to the detection module. The analysers or sensors which filter the information and discard irrelevant data are located inside the detection module. Finally, the data are sent to the response module. The response module decides if an alarm will be sent to the system administrator basing on predefined policies.

7.2.3 IDS based on autonomous agents

According to [62, 72, 30], the mobile agents are suitable to IDS since they offer scalability, resilience to failures, code independency, network traffic reduction, facility to perform previous proves to the agents in an independent manner before deploying them to the system, among others.

The architecture for IDS based on autonomous agents is built by the following components:

Monitors: They are data processing entities and the main controllers of the system. Monitors have an overall vision of the state of the network and can detect suspicious activities. They can also raise alarms and are hierarchically connected

to other monitors. In addition, monitors offer an interface that allows users to interact with the system.

Transceivers: They control all the agents in a host and can process data sent to them from the host. A transceiver communicates with the monitor on which it depends, within the hierarchical structure. Moreover, it can start, stop or eliminate the agents that are dependent on it.

Agents: They can be distributed at points within the network in order to monitor particular traffic in this network segment. An agent is a separate process that stores states, carries out simple or complex actions and exchanges data with other entities. Each agent generates a report and sends it to the transceiver but it cannot generate an alarm.

Filters: They make a selection of data and send the registers to the agents that correspond to the given selection criteria. There is only one filter for each data origin and the agents can be subscribed to some of them.

AAFID system [3] includes a user interface as a component of its architecture. User interfaces use APIs exported by the monitor, to ask for information and to provide instructions.

7.2.4 Risks in an IDS based on agents

The internal security of an IDS based on autonomous agents is an important factor to keep in mind, therefore it is necessary to protect the access to the platform and to the agents to ensure the privacy and the integrity of the data exchanged among them.

Although the mobile agents offer many advantages, because of their nature they also incur risks. Possible threats are the following: agent against the platform, platform against the agents, agents against other agents and other entities against the agent's system. There are different solutions to reduce these risks [63]. In this work we analyze the threats of the platform against the agents to offer a possible solution because they are the more difficult to prevent. This is because the platform has access to the data, code and results of the agents located on it. In this way, if a host is malicious, it can perform an active or passive attack.

In the case of a passive attack, the host obtains secret information as electronic money, private keys, certificates or secrets that the agent utilizes for his own requests of security. On the other hand, to perform an active attack, the host would be able to corrupt or to modify the code or the state of the agents. A malicious host can also carry out a combination of passive and active attacks, for example, by analyzing the operation of the agent and applying reverse engineering to introduce subtle changes, so

the agent shows malicious behaviour and reports false results. Our two proposals are focused on verifying the integrity of the agents, transceivers or monitors in runtime.

7.3 Mobile Agent integrity System - A security system to IDS based on autonomous agents

In an IDS based on autonomous agents, a monitor controls a network segment and it sends a transceiver to each host. Likewise, various agents are generated by a transceiver in order to monitor a determined type of traffic and they send alerts of suspicious activities to the transceiver on which they depend within the tree structure. One of the existing threats in these systems is when an intruder attempts to replace any IDS entity by another with similar characteristics but subtly modified in order to avoid a particular suspicious activity. So, if an agent or transceiver is modified or replaced, they will not report their correct results to their correspondent monitor and likewise, if a monitor is replaced it will not avoid or prevent the forthcoming attack.

Security solutions in IDS based on agents are the same that are offered for any environment that use agents. However, all the requirements are not covered; in particular, the threats against the IDS, its components and communications are not faced. So, in this section we propose to detect attacks against any IDS entity with a new security scheme named MAIS.

7.3.1 Scheme proposal

We propose a new system to verify not only the integrity of transceivers located in different hosts of the IDS architecture, but the correct execution of the transceivers during its operation. The MAIS system architecture is similar to AAFID system, but the transceivers and monitors behave like mobile agents and their mobility is limited, they only can displace to their corresponding trusted entity, that is to say, the upper level entity from which they depend. The data collection agents are static and they conserve the same characteristics of the AAFID system agents.

7.3.1.1 MAIS Architecture

The MAIS architecture has three essential components: monitors, transceivers and data collection agents. The monitors are agents that are located in the high levels of the infrastructure, they carry out correlation of information of high level and they control a network segment. There is a root monitor located in the higher level. It has the ability to communicate with an administrator interface and it also can provide the access point for the whole MAIS system. The administrator interface is independent of the IDS entities, in order to permit different implementations. The monitors can

also control other monitors and besides they are in charge of emitting and controlling another type of agents called transceivers. In MAIS, the monitors are also Trusted Parties, which are in charge of identifying the entities that they control and to carry out the process of watermarking recognition. The watermark allows us to verify not only the transceiver's or monitor's integrity but also their correct execution (a wrong execution generates a wrong watermark). Transceivers carry out correlation functions and they send the information to the monitor which they depend from. Transceivers have information about the host where they reside and also control the underlying agents. The main differences between an AAFID transceiver and a MAIS transceiver are the mobility and the mark. The data collection agents inside the MAIS infrastructure are in charge of monitoring a host and its behaviour. The agents and their transceivers are located in the same host. In the MAIS system, the transceivers and monitors must be mobile because they have to displace from its host to their TTP. This TTP is the immediately superior entity in the infrastructure, which will be able to do the mark verification; therefore, it is necessary to establish new characteristics for the system. The first one is that all the monitors and transceivers of the IDS must be mobile. The second one is that an entity which controls other entities must behave as a trusted party when thus be required, that is, to perform the mark verification. The third one is that each host must have at least two transceivers being able to carry out the same function, so when an agent is sent to the TTP to verify the integrity of its code and of its execution, another agent replaces its functionality. The transceivers depend on monitors and monitors likewise can depend on other monitors (figure 7.1), but the transceivers can only control their underlying static agents (data collection agents). So, the monitors are required to be trusted parties and they control the marking and verification processes to its underlying entities. The monitors have an overview of a network segment and the transceivers have an overview of a host.

7.3.1.2 MAIS system operation

The system operation protocol is as follows:

1. A monitor generates an entity and its corresponding support entity.

2. Subsequently it performs the watermarking process on each entity and sends them to the destination host conserving its timestamp.

3. The entities move to its destination host to carry out its function.

4. The agents periodically move to their generating entity according the established time (timestamp). While an entity goes to verify the integrity of its code and of its execution, the support entity continues carrying out their work.

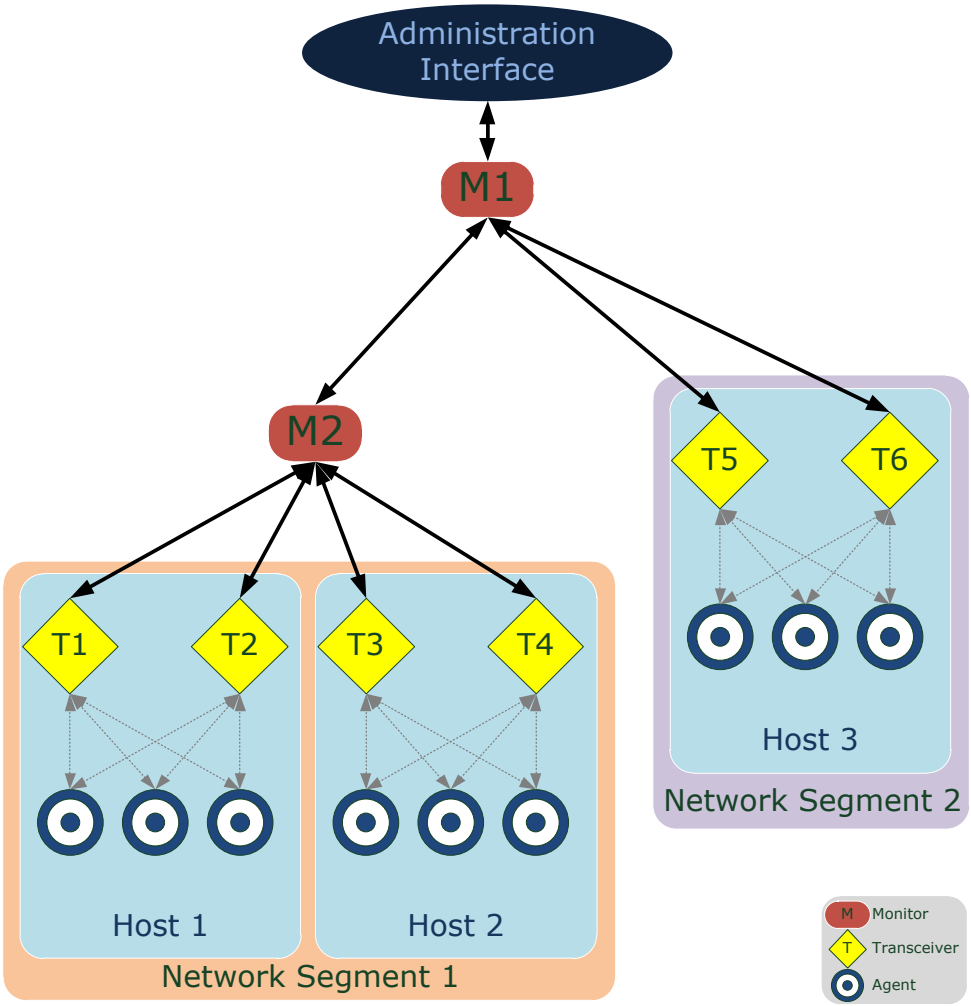


Figure 7.1: MAIS System Architecture

5. When an agent arrives to verify its integrity, the issuing entity performs functions of third trust party verifying the mark of the agent. If the agent has been compromised, the TTP eliminates it and isolate the host in which was residing, considering it malicious.

6. In case that an agent do not arrive on the established time to perform the verification process, the host is isolated and the agent is eliminated.

The issuing entity conserves the timestamp to verify that each agent arrives on time to control its integrity in determined periods of time. In each host there will be at least two entities executing the same function to provide service continuity while an entity displaces to carry out the verification of its integrity. When the agent arrives to the verifier, the mark is checked. Is important to note that an incorrect

transceiver's execution generates a wrong mark; so, the system administrator can detect the anomalous behaviour and perform the corresponding security measures. On the other hand, watermarking techniques are used instead of digest techniques because the transceivers are constantly being self modified to incorporate the new collected information.

7.3.1.3 Watermarking layer

The algorithm that we use to embed the mark is the Dynamic Graph Watermarking [23, 22] discussed in section 2.4, but others watermarking algorithms may be considered.

The main characteristic of the Dynamic Graph Watermarking algorithm is that it offers protection against distortive de-watermarking attacks as obfuscation or optimization. The basic idea is to embed the mark into a graph topology. If the agent is correctly executed, this graph is dynamically built during run-time; otherwise, a bad graph will be generated. As it is well known, dynamic graph structures are hard to analyse. On the other hand, semantic source code modification does not affect these algorithm performances because execution results must be the same, in other words, the agent has to generate the same graph structure of its watermark. From this point of view, the efforts to solve the watermarking problem will be concentrated in mark embedding and extracting methods.

Embedding process: The monitor selects a number n as product of two big prime numbers p and q. The number n is embedded in the topology of a graph by using Radix-K. After that, a source code which builds the graph is constructed. This code is embedded into the original agent producing the watermarked agent. When the watermarked agent is run with a fixed input, the graph will be built and the recognizer is constructed. The objective of the recognizer is to identify the graph on the heap of the agent execution. After that, tamperproofing and obfuscation techniques can be applied. Finally, the recognizer is extracted from the application and the watermarked agent is sent to its destination host. When a manipulated agent is moved to their generating entity, a monitor for instance, this entity can identify if the execution of this agent has been modified by linking this agent with the recognizer and executing them with a fixed input. As result, the modified watermark n' is obtained and this entity can verify that the original factors p and q can not factorize n' and it allows to detect the malicious agent. In other words, if n and n' does not match; the agent execution has been modified with high probability.

Mark extraction: As was commented before, the idea is to construct a graph in memory which topology embeds the mark. To recover this mark, an extraction process is needed. One method can be to examine all reachable heap objects but this can be a hard computational problem. Instead of this, the fixed input is divided in parts an

every part builds a portion of the watermark. As a result of the last part, the recognizer returns the root node of the watermark.

Watermarking justification: Digital signatures are widely used to guarantee the code integrity and authenticity. The digital signature can be used to verify, at a given moment, that a software code is exactly as created. However, it cannot assure that the code was properly executed over a period of time. In the IDS, given that the transceivers are changing continuously because they are collecting information, digital signatures techniques are inappropriate. Moreover we want to provide not only transceivers integrity but the correct execution of the transceiver. Therefore, we propose to use a watermarking technique which is suitable because the mark is dynamically built during run-time and if the semantic source code is modified the agent has to generate the same graph structure of its watermark, otherwise it indicates that the agent execution has been modified.

7.3.2 Discussion

The attacks of malicious hosts against the agents are considered one of the problems most difficult to solve and there is not a form of protection that eliminates them completely. To offer a determined security level in an IDS based on agents is necessary to combine different techniques that permit to detect an attack although it cannot be avoided. The drawback to send an agent to a malicious host is that this can be attacked, because of the host has total access to the code and data, therefore, to carry out a verification of its integrity, we propose the use of trusted monitors using watermarking techniques to verify the proper working of the IDS software components.

7.4 Improvement of Cooperative Itinerant Agents platform

A drawback of MAIS system is the fact that the agent has only been verified by a monitor so, the agent has to move on to the monitor. During this time, the platform needs a support entity which will assume the task of the agent during its verification. An improvement of this proposal is the use of a Cooperative Itinerant Agents in order to make some part of the verification in the same host in which the agent is executed.

The CIA security scheme, which consists in using itinerant cooperative agents, was proposed in [94] to verify the integrity of the monitors and transceivers in an IDS based on autonomous agents. In this section, the correct use of software watermarking techniques is discussed to achieve the desired security level required by CIA security scheme. In this scheme, each monitor in the system generates a transceivers agent for each host in the network segment that it controls. Similarly, the monitor embeds a watermark onto each transceiver and keeps a copy. The transceivers generate

information-collection agents that are located in the lower level of the IDS infrastructure. The monitor generates a cooperative-itinerant agent with a previously defined itinerary. Thus, the agent travels through the network segment that is controlled by the monitor that generated this agent.

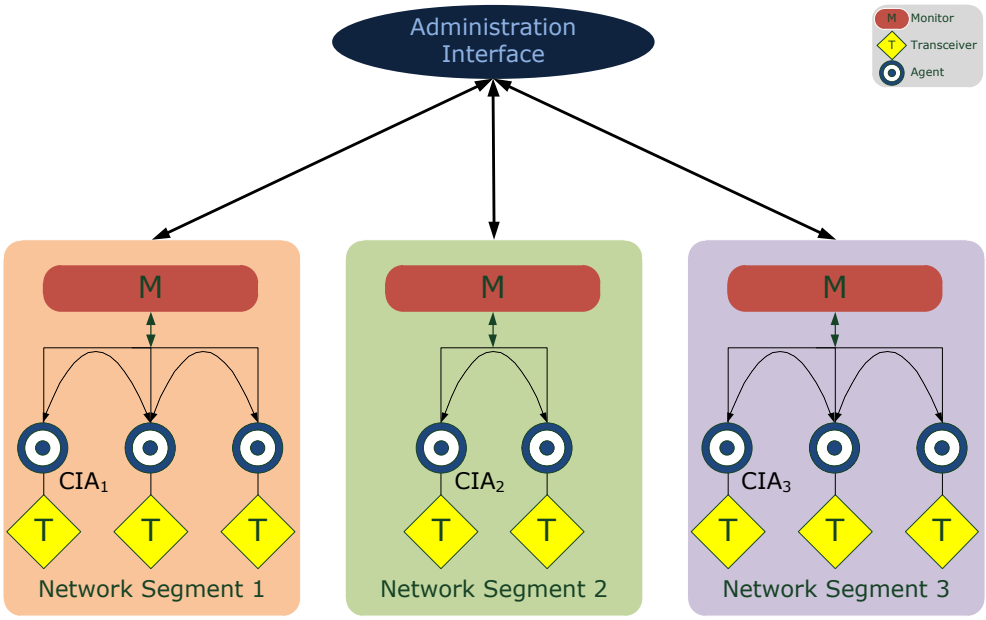


Figure 7.2: Transceivers verification by cooperative agents.

In figure 7.2, the process of CIA agents is illustrated. The network segment controlled by a CIA corresponds to the underlying level of an issuer monitor within the tree infrastructure. Every time that the CIA arrives at a host, it requests its fingerprint mark via the corresponding transceiver; subsequently the agent forwards the response to the monitor. The monitor verifies that the mark requested is correct by comparing it against each mark belonging to its set of marks. If the mark does not match any of the marks in the set, then it is assumed that the agent was manipulated, so the monitor will be able to act. This consists in eliminating the suspicious transceiver or isolating the malicious host.

The monitors' verification is performed in a similar way to that of the transceivers; when the CIA moves from one host to another, it reports its new destination to the monitor and continues its itinerary, repeating the process in each host. Each CIA can be configured to monitor entities with given profiles, for example, to verify transceivers located in a rank of directions or only to verify monitors. The agent can be programmed to cover either a previously defined route or a random one

A special case of verification occurs when monitors are located at a high level within the hierarchy but there is no upper entity to verify them. In this situation, a

cross verification of the monitors in the same level must be performed. This means that there must be at least two monitors in the root level. Each monitor within the upper level then generates a cooperative agent to verify the integrity of the neighbouring monitors.

7.4.1 Protecting agents against replay attacks

Because the malicious host has access to the code and state of the agent, the agent is exposed to such attacks, which is a disadvantage of the CIA security scheme. If the host performs an active attack, the monitor will be able to detect it because it will either not receive a notification on time or it will receive an incorrect notification. However, if the host performs a passive attack, it can, for example, detect and copy the response sent by the transceiver or CIA. Thus, when the agent is verified again, a malicious host can replace the response to deceive the monitor (i.e., perform a replay attack).

In order to avoid this kind of attack we propose using a particular mark. This mark is a matrix that identifies each monitor and transceiver in the IDS. The matrix is in turn split into various submarks; when a CIA arrives at a host it requests a set of submarks through a particular function. The corresponding entity (monitor or transceiver) responds with the result of another function. In this way, if a malicious host intercepts the communication, the information obtained cannot later be used to deceive the verifier. We also propose using software obfuscation techniques to prevent or hinder the process of reverse engineering or code analysis by a malicious host (see subsection 7.4.3).

7.4.2 Using a matrix of marks

Software can be identified by a mark; thus it is possible to prove not only its integrity but also to reclaim copyright. This mark should be embedded in a place known by the verifier and must seem like part of the results, that is, it should be imperceptible and resistant to transformation attacks. However, the mark should not influence in the operation of the marked code. The marks are static when they are stored in the application code and dynamic when they are constructed at runtime and stored in the dynamic state of the program [20, 21].

To identify transceivers and monitors from an IDS based on agents, we propose a matrix be used as a mark. The matrix has a fixed, previously determined dimension: $m \times n$. Each cell contains a prime value; prime values are considered submarks. To request the mark we use a cooperative itinerant agent (CIA). The CIA uses the following function to request a set of submarks from the IDS entity:

$$f_1(x) = \{(xp + r_1), (xq + c_1), (xr + r_2), (xs + c_2)\} \tag{7.1}$$

where x is an integer greater than or equal to zero and represents the module used by the agent; it masks the values and is known by the transceiver. The values p, q, r and s are random prime numbers that change every time the agent uses the function. The values (r_1, c_1) and (r_2, c_2) correspond to row and column numbers of two prime numbers of the matrix $(row_1, column_1, row_2, column_2)$, which are chosen randomly every time the agent attempts to verify a transceiver.

The submarks requested by the CIA are located in the cells (r_1, c_1) and $(r2, c2)$. The transceiver receives the four parameters and applies the corresponding modular reduction $(mod x)$ to obtain the coordinates of the matrix. In this way, the transceiver obtains the values w_1 and w_2 located in these positions and multiplies them. The transceiver applies the following function:

$$f_2(y) = \{yt + w_1 w_2\} \tag{7.2}$$

where t is a random prime number that changes every time the transceiver uses the function and y is an integer that represents the previously established module that will be used in the function. This module is known by the monitor and is used to mask the submarks. Thus, if the function's result is obtained by a malicious user, no information will be revealed. The CIA receives the result and forwards it to the monitor. The monitor applies the reverse process using the module y and compares this result against a second result. This second result is obtained by applying the same module to the result of $w_1 w_2$. If the operation does not match, it means that the transceiver has been modified and should therefore be considered malicious. Otherwise, it is highly probable that the agent has not been modified.

7.4.2.1 Example

Next, the process of marking a transceiver will be explained. The first step is to issue a matrix to be used as a mark with a size calculated by $m \times n$ (each matrix is unique to each transceiver). Each cell contains different prime numbers w_j and these are considered submarks. A module x must be assigned to be used with equation 7.1, which is fixed during the process. In contrast, various random values must be issued by the monitor each time the CIA begins the verification process; these values are issued to multiply the module and to select two positions in the matrix. The matrix of marks and values are shown in tables 7.1, 7.2 and 7.3.

Matrix of marks: Firstly, a monitor generates a transceiver and assigns a matrix to mark it. The monitor stores a copy of the matrix. The matrix remains fixed and when a CIA arrives at a host, the monitor gives it the masked coordinates, where the values to be verified are located.

	0	1	2	3
0	68813	79687	36599	98663
1	59879	16993	98689	36997
2	79657	11383	35729	21991
3	78643	41299	86323	59693

Table 7.1: Matrix of marks

Fixed values used in the process: After that, the modules that will be used by functions f_1 and f_2 have to be fixed.

Variable	Value	Description
x	17	Module of the function $f_1(x)$
y	53	Module of the function $f_2(y)$

Table 7.2: Fixed values used in the example.

Random values issued every time that the CIA arrives at a host: the values $(r_1, c_1) = (0,2)$ and $(r_2, c_2) = (2,3)$; correspond to the positions of the values 36599 and 21991 in the matrix of marks that we are using.

The CIA agent utilizes the function 7.1:

$$\begin{aligned} f_1(x) &= \{x \cdot p + r_1, x \cdot q + c_1, x \cdot r + r_2, x \cdot s + c_2\} \\ &= \{17 \cdot 37 + 0, 17 \cdot 13 + 2, 17 \cdot 7 + 2, 17 \cdot 23 + 3\} \\ &= \{629, 223, 121, 394\} \end{aligned}$$

The transceiver utilizes the values and applies the reversed process to obtain the positions of the matrix where are located the requested values:

$$\begin{aligned} C &= \{629, 223, 121, 394\} \mod x \\ &= \{629, 223, 121, 394\} \mod 17 \\ &= \{0, 2, 2, 3\} \end{aligned}$$

These values correspond to the row 0, column 2 and row 2, column 3 of the matrix; in these positions are located the values 36599 and 21991. The transceiver utilizes the function 7.2. We are going to use $y = 53$ and $t = 3571$.

Variable	Value	Description
p q r s	37 13 7 23	Random values, which multiply to the module of function 7.1.
t	3571	Random values, which multiply to the module of function 7.2
r_1	0	Row of the matrix where the first submark (w_1) to be verified is located ($r_1 < m$: where m corresponds to the number of rows in the matrix).
c_1	2	Column of the matrix where the first submark (w_1) to be verified is located ($c_1 < n$: where n corresponds to the number of columns in the matrix).
r_2	2	Row of the matrix where the second submark (w_2) to be verified is located ($r_2 < m$: where m corresponds to the number of rows in the matrix).
c_2	3	Column of the matrix where the second submark (w_2) to be verified is located ($c_2 < n$: where n corresponds to the number of columns in the matrix).

Table 7.3: Random values issued every time that the CIA Matrix of marks

$$\begin{aligned} f_2(y) &= \{t \cdot y + (w_1 \cdot w_2)\} \\ &= \{3571 \cdot 53 + (36599 \cdot 21991)\} \\ &= \{805097872\} \end{aligned}$$

The CIA applies the reversed process and obtains the module of the received value:

$$\begin{aligned} S_1 &= f_2(y) \mod y \\ &= 805097872 \mod 53 \\ &= 43 \end{aligned}$$

Subsequently, it obtains the module by multiplying the requested sub marks:

$$\begin{aligned} S_2 &= 36599 \cdot 21991 \mod y \\ &= 804848609 \mod 53 \\ &= 43 \end{aligned}$$

Afterwards, the CIA verifies that the module sent by the transceiver is equal to the module of the result obtained by multiplying the requested values (36599, 21991), that

is to say, $S1 = S2$. In our example, the comparison is correct, so there is a high probability that the transceiver's execution has not been modified.

In this example, we are using a matrix with a size of four rows by four columns; in this way the possible combinations of obtaining the same pair of sub marks is given by:

$$\frac{n!}{w!(n-w)!} = \frac{16!}{2!(16-2)!} = 120$$

Each time that the CIA goes to verify the integrity of a transceiver, it chooses randomly a set of different sub marks and ask for them to the transceiver or monitor. Thus, with a matrix with size of 16 positions and requesting only two marks, a CIA agent has a low probability of request the same pair of values (1/120). By simplicity, we have chosen as mark a matrix with 16 positions and small values to fill the matrix and to utilize as module in the functions 7.1 and 7.2. Likewise, in this example, only two sub-marks have been chosen to verify the transceiver's execution, but there is the possibility of request more sub-marks; in this case, the CIA must provide the different positions of the values to be verified. On the other hand, the transceiver must utilize the values in function 7.2 and perform the multiplication among them. In spite of using a module's function to mask the results, in order to avoid that a host malicious can deduce the operations performed by the transceiver or monitor; there is a possibility that the sub-marks can be detected when a CIA request it (although the probability of requesting the same marks combination is very low). This drawback can be solved using sub-marks of single use, that is to say, each time that a sub-mark is requested, it must be blocked when the verification process finish. So, the CIA does not request a mark twice, and when all the sub-marks of a determined entity have been requested by a CIA, it notifies to the monitor. So it will issue another transceiver with its corresponding matrix of marks to replace the previous. This is an additional measure to avoid the analysis and possible detection of the sub-marks by a malicious host. The following table shows the possible combinations with matrixes of 16, 32, 64 and 128 positions choosing 2, 3 or 4 sub-marks to verify the integrity of a transceiver.

# of positions	2 sub-marks	3 sub-marks	4 sub-marks
16	120	560	1820
32	496	4960	35960
64	2016	41664	635376
128	8128	341376	10668000

Table 7.4: Matrix Positions *versus* set of possible sub marks to verify.

According to the table 7.4, using a matrix with size of 16 positions there are 120 possible combinations of choosing two different sub marks to verify them; if the ma-

trix has 128 positions, the maximum number of combinations is 8128. Requesting the same number of sub marks but with matrixes of different size, the combinations number grows significantly. It is important to note that depending on the number of requested sub marks and the size of the prime numbers (modules, numbers of the matrix, multipliers random numbers), the function's results used by the transceiver can be too large and it would induce an overflow error.

The previous verification process of a transceiver or monitor can be considered as a Zero-Knowledge Proof (ZKP). The ZKP is an interactive protocol between two parts, the part which provides the proof (*prover*) and the part which verify it (*verifier*). The *prover* must convince to the *verifier* that he knows the solution of a theorem given without revealing any type of additional information [100]. The *verifier* can ask to the *prover* in several times about the solution and these questions are different and random in order to avoid a sequence; thus, is not possible to memorize it. Likewise, if an attacker intercepts the exchanged messages between *prover* and *verifier*, no confidential information is revealed to him.

In our case, the transceiver must solve the function 7.2 with the sent parameters by the CIA, and on the other hand, the CIA must verify if the transceiver's response is correct. There is a low probability of guessing correctly the response without knowing the protocol operation and the operations of the corresponding functions. Moreover, each time that the CIA goes to verify an entity's integrity (transceiver or monitor), it sent different and random matrix positions; in this way it is not possible to deduce a sequence in order to memorize it and to perform a posterior attack; thus it reduces the deceive probability. Another additional measure to protect the IDS's entities integrity from malicious host is using obfuscation techniques to avoid code analysis or reverse engineering.

7.4.3 Code obfuscation

Code obfuscation is a technique utilized for altering the structure of a program to difficult its reading and thus harder to understand its operation for unauthorized users. From the computer's point of view, it is simply a translation to be performed and the compiler can process easily the obfuscated code. To do code obfuscation, it is necessary to use an obfuscator program, which transforms the application in another that is functionally identical to the original. The obfuscator program inserts irrelevant code into loops, unnecessary calculations, performs data consult that will not be used, etc. This technique is suitable to protect secret marks but it does not can avoid reverse engineering, because a programmer with enough knowledge and time could recover the algorithms and data structures of the analyzed code [21]. In this case, the strategy

is to discourage unauthorized entities doing the information analysis more expensive than the information itself. There are several algorithms to perform code obfuscation [21, 76] and they can be applied to the static and mobile agents. In this way, if a malicious user analyzes the code of an obfuscated agent, it will not be able to understand it easily or simply, the process will be so expensive. Moreover, if an attacker obtains the used functions or the values returned by an agent, this information will not reveal important data because the functions results are masked by different modular functions.

7.4.4 Mark embedding

The algorithm used in order to embed the mark was proposed in [87] (see section 2.4). This algorithm contributes with copyright protection and it is a tool to distinguish each copy of specific software, in other words, it provides the system with fingerprinting properties. This algorithm is based in the use of branch functions to generate, in run time, the appropriate fingerprinting. The original formulation can be summarized with these two processes (embedding and extracting process):

$$embed(P, AM, key_{AM}, key_{FM}) \Rightarrow P', FM$$
$$recognize(P_0, key_{AM}, key_{FM}) \Rightarrow AM, FM$$

The first process requires as input the program (P), the Authorship Mark (AM) and copyright and fingerprinting keys. Note that, while the copyright key will be the same for all copies, the fingerprinting key will be different for each of them. As result of this process, the correctly marked program (the agent in our scenario) and the corresponding fingerprinting code are obtained. On the other hand, the recognize function have as input a piece of marked code (or a function of the program) and 2 keys. As output, this function retrieve the Authorship Mark (AM) and Fingerprint. In the scenario presented in this section, this algorithm is used in order to embed the matrix presented in the section 4. The aim of the mark embedding is to obtain a system that retrieve a value from two input values (in our case, obtain a value in the matrix from values r and c). In the embedding process we can use r as key_{AM}, c as key_{FM}, and P as a pointer to piece of source code that will embed the value in the row r and in the column c in the marks' matrix. Note that the embedding process must be done for each value in the matrix. In the same way, the recognize function will be adapted. The new formulation can be summarized with these two processes (embedding and extracting process):

$$embed(P, AM, r, c) \Rightarrow P', M(r, c)$$
$$recognize(P_0, r, c) \Rightarrow M(r, c)$$

where $M(r,c)$ is the value in the row r and column c of the marks' matrix and P and P_0 are pointers to a piece of the program.

7.4.5 Discussion

The security of the agents in an IDS can be faced in a particular way, taking advantage of the tools provided by the environment, because in this case the system permits to issue verification agents through the upper entities in the tree infrastructure. In this way, using fingerprinting and watermarking, the risks of attacks from malicious host can be reduced. Likewise, code obfuscation is an additional tool which can be used to avoid code analysis and reverse engineering or to protect the entities' integrity from malicious host.

7.5 Conclusions

On one hand, attacks from malicious host against agents are considered one of the most difficult problems to solve and there is not a way to avoid them. On the other hand, the use of mobile agents in IDS is increasingly common so, in order to offer the required security level in IDS based on agents, it is necessary to combine different techniques to detect a possible attack although it can not be avoided. The drawback of sending an agent to a host is that the host could be malicious and it can attack the agent because it has whole access not only to the agent data but the code. Two different alternatives based on the use of software watermarking techniques are presented in this chapter. The first one is the use of 2 entities for each network segment which control the activity of this network. After a fixed period of time, one of these entities is moved on to the monitor in order to be verified. The second proposal uses a cooperative itinerant agent to verify into the monitored host the agent in charge of a network segment. This system uses a matrix of marks and it determines if an agent was modified by asking subsequently a set of sub-marks. Moreover, in these proposals, tamperproofing and obfuscation techniques are used to make harder the work of modifying an agent execution.

CHAPTER 8

PROTECTION OF MOBILE-AGENT EXECUTION USING A MODIFIED SELF-VALIDATING BRANCH-BASED SOFTWARE WATERMARKING WITH EXTERNAL SENTINEL

Critical infrastructures are usually controlled by software entities. To monitor the correct functioning of these entities, a solution based in the use of mobile agents is proposed. Some proposals to detect modifications of mobile agents, as digital signature of code, exist but they are oriented to protect software against modification or to verify that an agent has been executed correctly. The aim of our proposal is to guarantee that the software is being executed correctly by a non-trusted host. The way proposed to achieve this objective is the improvement of the Self-Validating Branch-Based Software Watermarking by Myles *et* al. [87]. The proposed modification is the incorporation of an external element called **sentinel** which controls branch targets. This technique applied in mobile agents can guarantee the correct operation of an agent or, at least, can detect suspicious behaviours of a malicious host during the execution of the agent instead of detecting when the execution of the agent have finished.

8.1 Introduction

Nowadays, software entities control most of the critical infrastructures. Usually, these systems must have an accurate exactitude due to the importance of its task but, what happens when these systems are compromised? Some mechanisms are applied in or-

der to protect or increase the security level of these kinds of infrastructures. Commonly used systems like firewalls, intrusion detection systems, honeypots or honeynets,... are useful to detect and counteract security incidents but, these methods are as necessary as the systems which control that these security systems work correctly. It seems logical that these systems must be constantly monitored.

The logs were the first tool typically utilized to monitor systems but its have turned a slow and not enough real-time monitoring system. Over last years, the systems based on alerts have been presented as the alternative because its can detect anomalies in the systems and send alerts to the central control to inform the manager. In this scenario, mobile agents can be a good solution because its use implies low resources wasting from the point of view of central control since it is the monitored system which provides these resources. In other words, an agent is sent to a critical infrastructure to control the software which manages this infrastructure. If there are not security incidents, the agent will not send information to central control and, as a consequence, it will not spend bandwidth or CPU time of this central control. Otherwise, if a security incident is detected by the agent, it will send the relevant information about the security incident to central control.

Since the idea of mobile agents appeared, some related security challenges have arisen. These security challenges can be basically divided into 2 major groups: how to protect a host from a malicious agent and how to protect an agent from a malicious host. Some proposals try to solve the first problem. In this case, these proposals guarantee that the agent which will be executed is the agent which the host thinks. In other words, these are *a priori* security systems. On the other hand, the proposals which try to solve the second problem are *a posteriori* protection system. That is to say, these systems are methods which can use the origin host to verify that the host which have been executed the agent have been proceed honestly. The aim of the system presented in this paper is to guarantee the well-function of an agent in a malicious host or, at least, to detect that something is adulterating the normal operation of this agent. The main novel contribution is that this detection is done while the agent is being executed by the malicious host. The technique presented is based in an original use of watermarking mechanism and the use of branch functions. The algorithm presented is a modification over the algorithm presented by Myles *e*t al. in [87]. The designed modification is the inclusion of a **sentinel** element which controls the targets of the branch functions.

In the following section, the basic concepts are introduced. Section 8.3 explains the modifications done to the Self-Validating Branch-Based Software Watermarking by Myles *e*t al., presented in section 2.4.4, in order to protect mobile agent execution. Section 8.4 deals with the security aspects. Some implementation aspects are commented in section 8.5. Finally, some conclusions are given.

8.2 General Concepts

Some general concepts used below will be introduced in this section. A brief explanation about how each technique is used in this proposal is also given. Software agents can be defined as software entities that migrate between hosts in order to obtain data or perform operations autonomously of user action. Taxonomy of the existent agents can be found in [40]. The main attack to mobile agent environments is performed when the host that executes the agent carries out a malicious action in order to produce a dysfunction of the agent.

One of the used technique is software watermarking. The aim of digital watermarking techniques is to embed information into any digital content. The main characteristic is that the embedded information will be imperceptible for the users of this digital content. A taxonomy of software watermarking and knew attacks was presented by Christian Collberg and Clark Thomborson in [22] and [23]. In this approach, a watermarking technique is used in order to insert information by means of an imperceptible mode into the code of the agent without affecting the normal behaviour of the agent.

In parallel to watermarking techniques, there exist similar techniques which are known as fingerprinting techniques. The concept of fingerprinting was introduced by Wagner in [116] as a method to protect intellectual property in multimedia contents. The fingerprinting technique consists in making the copies of a digital object unique by embedding a different set of marks in each copy. In this approach, this technique is used in order to produce different and differentiable copies from the same agent. In this way, the agents can be reused several times by means of watermarking copies of the same agent with different fingerprints.

8.3 Self-Validating Branch-Based Software Watermarking with external sentinel

As has been introduced in section 2.4.4, the authors of SVBBSW algorithm, Myles et al., defined a special type of branch functions that is named Fingerprinting Branch Function (FBF). In addition to normal branch functions behaviour, this kind of function modifies the k_i value, which is an integrity check of some part of the code of the agent, every time that this function is called. This value is used to obtain the destination of the branch in each iteration. Moreover, in the last iteration, k_n will take as value the fingerprint (FM). Taking into account its functionality, the original Fingerprinting Branch Function can be divided in three different modules:

- k_i **calculator**: the aim of this module is to calculate k_i with any prefixed one-

way function which involves k_{i-1}, an code integrity check v_i and authorship mark AM.

- **Mapping** between the different values of k_i and the pointers to next instruction to execute in the program execution flow.

- **Execution control transferrer**: This part is in charge of redirecting the program execution flow to the target instruction that is pointed out by the pointer which has been supplied by the previous mapping.

In the original algorithm, these three parts were indivisible and its stayed in the same function which did these three operations. In this proposal, these parts are differentiated and divided. The k_i **calculator** and the **Execution control transferrer** will be in the same function but the **Mapping** is moved from the mobile agent to another host that is named **sentinel** so, the agent, has to request these information to this external entity in order to continue its execution flow.

The main difference between the typical embedding process of Myles *e*t al. algorithm and the proposed embedding process is in the third step. In the modified algorithm, the mapping is not included into the code. This information is located into the **sentinel** host instead of embedding it into the code.

The k_i **calculation** module is invoked when the Modified Fingerprinting Branch Function (MFBF) is called. This process is the same as the original algorithm and it must also depend on the code integrity check v_i , the previous value of k_i and the authorship mark (AM). The obtained value is used to find the next instruction in the normal execution process of the agent. This value is used by the **Execution Control Transferrer** module to do a request to the **sentinel** about the pointer to the next instruction. This request is sent through a secure channel between the audited host and the **sentinel** (see subsection 8.4 for the secure channel justification). The **sentinel** uses the information contained in the request as the index to search in the **mapping** between the possible values of k_i and its corresponding pointer. When the pointer is found, it is sent through the same secure channel to the **Execution Control Transferrer** and this module transfers execution control to the instruction indicated by the pointer. A more graphical explanation can be found in figure 8.1.

The **sentinel** can be a third trusted part but it can be adapted to a hierarchical structure in a mobile agent architecture which controls critically distributed infrastructures. In this case, the **sentinel** is the server in charge of controlling some part of the infrastructure and it can be simultaneously audited by another mobile agent with another sentinel as is shown by figure 8.2.

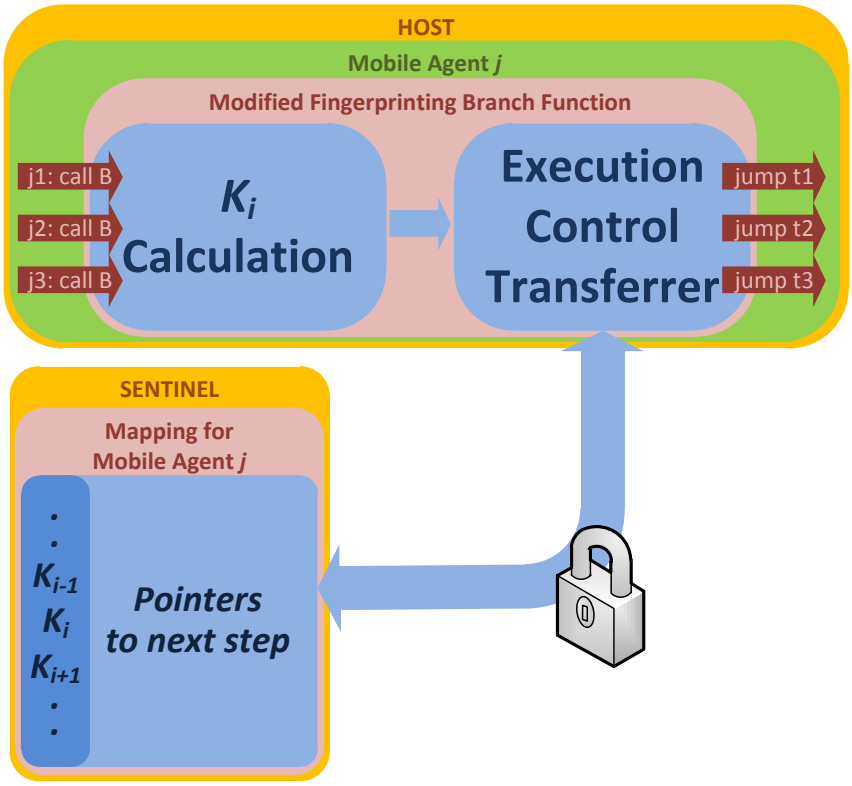


Figure 8.1: Schema of self-validating branch-based software watermarking with external control operation.

8.4 Security analysis

Note that the proposed system assumes that exists a secure channel from the monitored host to the **sentinel**. This is because assuming that a mobile agent can built a secure channel to an external host (the **sentinel** in this case) it is very unrealistic. In this way, the security analysis is centred in the vulnerabilities between the secure channel and the agent, in other words, the main problems against the system proposed are generated when the host which lodges the agent makes actions in order to produce dysfunctions in the correct execution of the agent. In this way, it seems a realistic approach to consider that it exists a secure channel between the monitored host and the **sentinel** because, in a critical infrastructure, the security of the communications between the different parts have to be guarantee.

Next, some aspects related to the security are commented:

k_i interception or modification: The k_i values can be intercepted and modified by the host when the agent sends its to the **sentinel**. If the host only performs a

passive attack, it can not obtain any critical information from this values so k_i is the result of one way function. In this case the **sentinel** never can detect this action. If the host performs an active attack, that is to say, k_i value is modified by the host, the **sentinel** can identify this action as an attack and will start actions to punish this host or simply isolate the part of the infrastructure that depends of this agent. In both cases, an alarm will be generated.

Returned pointer interception or modification: The returned pointer to the mobile agent from the sentinel can be intercepted or modified. The first case can not be detected by the **sentinel** but this action does not have dangerous consequences to the system. The modification of the returned pointer can cause a dysfunction in the agent but this dysfunction will produce an erroneous k_i value or will stop the agent execution. In both cases, this action will be detected by the **sentinel** with the new request from the agent or by an execution time control system like proposed in [58] or in [34].

***loop-process* attack:** The malicious host can try to perform a *loop-process* attack, that is to say, sending always the same k_i to the **sentinel**. As k_i is calculated taking into account the value of k_{i-1}, loop processes to confuse the sentinel are not possible because the one way function which takes part in k_i calculation will never repeat the value so, in a natural loop processes, the agent will produce a new k_i every time. In the same way, if the host try to perform a deny of service attack, this will be detected by the **sentinel**.

Debugging or disassembly: This algorithm includes code verifications and integrity

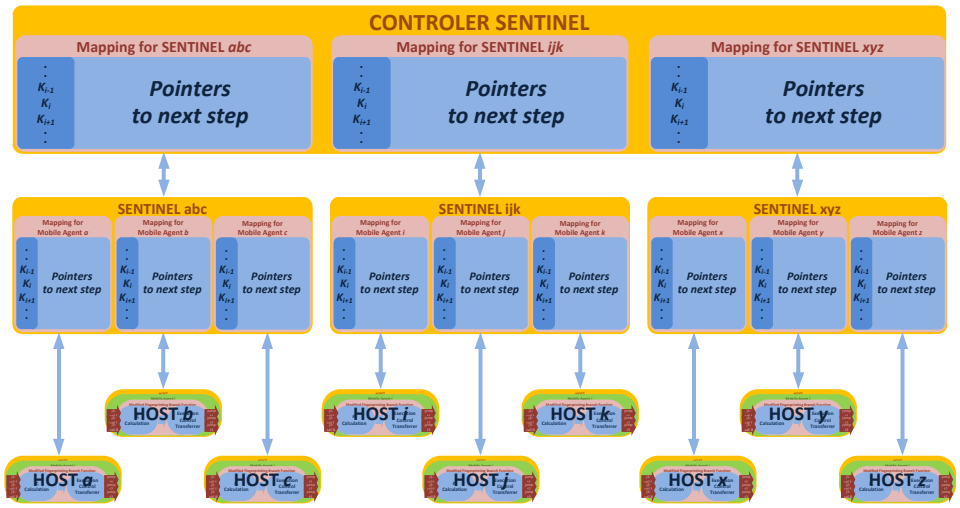


Figure 8.2: Control of critically distributed infrastructures with mobile agents and different `sentinels`.

checks in order to identify if the mobile agent has been subjected to semantics-preserving transformation or even if a debugger is present. Moreover, the branch functions are the typical solution against static disassembly of native executables so the malicious host is not able to perform this kind of attacks without to be detected by the **sentinel**.

Break or stop agent execution: These actions can be detected with execution time control system. Exist a lot of proposal about this techniques as [58] or [34].

Mobile agent renovation: Renewing periodically the agent by another copy of the same agent but with another fingerprint can be extremely recommendable because the new agent will have new k_i values which will be different from the old agent values. This action adds security to the system to prevent that the host can obtain enough information from the agent to perform a success attack to the system. This functionality has been taken into account in this proposal and it is easy by means of changing key_{FM} (see section 2.4.4).

8.5 Implementation aspects

There are some aspect that must be taken into account before apply this proposal in a practical system. These aspects may determine or can influence the design.

Off-line and autonomous execution: In the proposed system, the mobile agent needs a periodically connection to the sentinel in order to continue with its normal operation. But this is not a problem in a critical infrastructures control because the intercommunication between the controlled system and the controlling systems is always necessary. Nevertheless, the amount of bytes that the systems interchange can be minimised with this proposal.

Latency: Another aspect is the latency which is caused by the transmission time of requests and responses, from agent to sentinel and vice versa and the process time that is spent by the sentinel. Fortunately, the speed of the critical infrastructures networks is the appropriate and this time would be usually considered negligible.

Execution time: Obviously, when the number of calls to MFBF increases, the execution time also increases. So a threshold must be defined at the design time by the required security level and the maximum delay which the monitored system can tolerate between the moment when the incident occurs and the moment when it is detected.

8.6 Conclusions

A novel approach to guarantee the security of the mobile agents execution in a compromised host is presented. This approach is based in the algorithm presented by Myles *e*t al. in [87]. This algorithm has, as well as authorship protection capacities, integrity code check, tamper detection and fingerprinting capacities. The original algorithm is based in branch functions that transfer the agent execution to the correct point in the program execution flow from a calculation obtained of one way function which depends on an authorship mark and an integrity check. Our improvement is the incorporation of an external element called **sentinel** which keeps the relation or mapping between the values obtained from the one way function (k_i) and the pointer to the next instruction in the program execution flow. The controlled host needs to send the value of the appropriate k_i to the **sentinel** and waiting for the response to execute the agent in the correct way. On the other hand, the **sentinel** can control periodically the agent execution because it knows the correct sequence of k_i for each agent. Additionally, a security analysis of the proposed schema is presented and some aspects related to its implementation are commented.

CHAPTER 9

AN INFRASTRUCTURE FOR DETECTING AND PUNISHING MALICIOUS HOSTS USING MOBILE AGENT WATERMARKING

Mobile agents are software entities consisting of code, data and state that can migrate autonomously from host to host executing their code. In such scenario there are some security issues that must be considered. In particular, this chapter deals with the protection of mobile agents against manipulation attacks performed by the host, which is one of the main security issues to solve in mobile agent systems. An infrastructure for Mobile Agent Watermarking (MAW) is introduced in this chapter. MAW is a lightweight approach that can efficiently detect manipulation attacks performed by potentially malicious hosts that might seek to subvert the normal agent operation. MAW is the first proposal in the literature that adapts software watermarks to verify the execution integrity of an agent. The second contribution of this chapter is a technique to punish a malicious host that performed a manipulation attack by using a Third Trusted Party (TTP) called Host Revocation Authority (HoRA).

9.1 Introduction

Mobile agents are software entities that consist of code, data and state, and that can migrate from host to host performing actions on behalf of a user. Chapters 7 and 8 have dealt with the protection of mobile agents which are utilized for supervising Intrusion Detection Systems or to monitor critical infrastructures. This chapter will tackle the use of mobile agents as it is described in chapter 2, that is to say, when a

user wants to perform some tasks using mobile agents. Section 2.5 presents as example a user who wants to arrange a meeting with other people. In this kind of situation, mobile agents are used for obtain information from others hosts, which have the information of the people invited to the meeting, in order to find the best suitable option taking in account the restrictions specified by the user who have launched the agent and the availability information of guests contributed by others hosts.

In this chapter, the detection and punishment of manipulation attacks performed by malicious hosts are addressed. A malicious host performs a manipulation attack when, trying to achieve a certain purpose, modifies any part of the mobile agent to disrupt its proper execution. In the previous example, a possible manipulation attack could be that a malicious host modifies the execution of the agent to impose the date of the meeting without taking into account the agendas of the rest of invitees. As previously mentioned, a pessimistic view is assumed, considering the execution of the agent in a non-trusted community of hosts. In case that the hosts can be considered trusted, no protection mechanism is needed against manipulation attacks. The aim is to avoid manipulation attacks by dissuading hosts, because detection can lead to punishment. Taking into account these objectives, this chapter explains two mechanisms that work together to achieve an effective and usable protection mechanism against manipulation attacks.

In first place, an infrastructure for Mobile Agent Watermarking (MAW) is proposed. MAW is a lightweight approach to detect manipulation attacks. MAW is the first proposal in the literature that adapts software watermarks to verify the execution integrity. It must be clarified that the primary goal of MAW is not to develop a new software watermarking scheme but to use and adapt existing watermarking techniques to detect manipulation attacks against mobile agents. The novelty of this proposal is that the watermarks are used and embedded in a different way and for a different purpose than traditional software watermarking systems. Indeed, different types of watermarking techniques can be used in the presented infrastructure, and these techniques might also be changed in the future according to advances in the watermark research area.

The second contribution of this proposal is a mechanism to punish the malicious hosts by using a Third Trusted Party (TTP), that is, a trusted entity for all the entities of the system. This TTP is called Host Revocation Authority (HoRA from here on). The HoRA stores in a database the information of those hosts that have been proven malicious in order to avoid new attacks from them. The punishment mechanism proposed is based on the idea of host revocation, which essentially consists in avoiding sending mobile agents to the hosts that previously attacked other agents. Both detection and punishment working together can achieve an effective protection mechanism against manipulation attacks.

In this chapter, the guidelines about how to choose the watermarks to embed into the code of the agent are explained. From this point of view, this chapter introduces a new application of watermarking that may open a new research area. The rest of the chapter is organized as follows: Section 9.2 provides the reader with a quick review of the required background which is explained in depth in chapter 2; section 9.3 explains how to detect manipulation attacks using MAW; section 9.4 details the HoRA functionalities and, finally, the conclusions of this proposal can be found in section 9.5.

9.2 Background

This section summarizes some background topics that are necessary to easily understand our proposal. First, we review the problem of the malicious hosts and some of the published countermeasures to alleviate it. In particular, we emphasize the cryptographic traces approach [112] because we will compare it with our attack detection proposal MAW. Next, the main objectives of software watermarking schemes are summarized. Special attention has also been paid to the Collberg-Thomborson (CT) algorithm [23] because we use it in our implementation.

9.2.1 Malicious Hosts

The problem of malicious host has been addressed before in section 2.5. As it has been commented, the literature about countermeasures for malicious host attacks can be divided in two kinds of approaches: attack avoidance and attack detection approaches. Regarding attack avoidance approaches, they try to avoid attacks before they happen [119, 79, 58, 99]. However, published attack avoidance techniques are difficult to implement or computationally expensive. For this reason, it is considered that attack detection techniques are more promising because they are usually easier to implement. The objective of attack detection approaches is detecting manipulation attacks. One of the main representative approach is the use of cryptographic traces as propose Vigna in [112]. Vigna introduces the idea of the cryptographic traces, which are logs of the operations performed by the agent. The operations of the agent can be categorized in white statements, which modify the state of the agent due to internal variable values; and black statements, which alter the agent's state due to external variables. These traces contain the changes performed to internal variables as a consequence of black statements. A re-execution of the agent can be performed with these traces. Instead of sending the traces, the hosts must store them to save network bandwidth. This is due to their size depends on the amount of input data, which can be huge. If the origin host suspects that a host modified the agent and wants to verify the execution, it asks for the traces and executes the agent again. If the new execution

does not agree with the traces, the host is cheating. The approach not only detects manipulation attacks, but it also proves the malicious behavior of the host. However, this approach has two main drawbacks: (1) verification is only performed in case of suspicion, but the way in which a host becomes suspicious is not explained, and (2) for an indefinite period of time, each host must reserve enough capacity to the storage of traces of past transactions because the origin host can ask for them. These drawbacks can be relieved by controlling the execution time of the agent in the hosts [33], but even with this complement, the use of traces might be still too expensive for all the entities involved.

9.2.2 Software Watermarking

Software watermarking was designed to protect software copyright. In this sense, a watermark is embedded into software in such a way that, a trusted third party could check that this software belongs to a company. However, the mechanism presented in this chapter uses software watermarking techniques not to protect the copyright of programs, but to protect the execution of a mobile agent in an untrusted host, the Mobile Agent Watermarking (MAW) infrastructure. The objective of the owner of an agent is to assure that this agent has been properly executed by embedding a software watermark in the code of the agent. On her side, the objective of malicious hosts is also different, modifying the execution of the agent to obtain a certain profit. As we will discuss later in Section 9.3.5, attacks based on semantic preserving program transformations against watermarked agents are useless for the attacker in this scenario. This is because these transformations only affect the code appearance, not code behaviour. In this case, the modified code will be executed in the proper manner, so the execution integrity is assured. In this chapter, the design of the MAW proposal is also presented. It is based on a particular data structure watermark, the Collberg-Thomborson (CT) algorithm [23], which is also known as Dynamic Graph Watermarking. This algorithm is based on embedding watermarks within the topology of graphs built dynamically in memory during the execution of a program. How works the CT algorithm is summarized in section 2.4. The next section will address the concrete CT-based modification of MAW.

9.3 Mobile Agent Watermarking (MAW)

MAW is a lightweight approach to detect manipulation attacks. That is to say, with MAW one can verify whether an agent was or was not properly executed by a host. It must be stressed that this infrastructure is the first approach that adapts software watermarks to solve the problem of the malicious hosts.

The MAW infrastructure works as follows: the original agent is modified by introducing a dynamic software watermark. Watermarked agents generate output data according to a set of rules. These rules are named "*integrity rules*". Integrity rules are secret, that is to say, they are only known by the origin host. The watermarked agent generates output data organized in what "*data container*". The organization of the data container is performed according to the integrity rules. In this sense, it is said that the watermark is transferred to the data container. For instance, the process could be illustrated with one very simple integrity rule. It could be assumed that the agent generates an integer as a piece of output information. To introduce the integer in the container, it is multiplied by two if a certain global variable is even. The associated integrity rule is to check that this piece of data in the container is even if the global variable is even. In general, integrity rules are related with input data, internal values (heap, stack etc.), dummy data and data from external communications. Therefore, it can say that this proposed infrastructure is based on dynamic software watermarking (since watermarks are created at runtime depending on the state of the program).

Finally, the origin host receives the agent, which has been executed by all the hosts of the itinerary. Then, the origin host applies the set of integrity rules to all the data containers (there is one container for each host in the itinerary). These integrity rules are a set of logical properties that a container must fulfill (if it has not been tampered). These rules are responsible for demonstrating that the presence of the watermark is the result of deliberate actions. If a container does not fulfill the rules, this means that the corresponding watermark has been modified, and hence the corresponding host is malicious.

In summary, the process has three phases: (1) watermark embedding: modify the agent to embed the watermark generation code (see section 9.3.1); (2) watermark transference: create the container during the agent'ss execution to transfer the watermark and hide the results (see section 9.3.2); and (3) detecting manipulation: watermark verification using integrity rules (see section 9.3.3).

9.3.1 Watermark Embedding

Current software watermarking techniques must be adapted to our scenario because they were not originally designed for creating execution integrity proofs (containers). As explained in section 2.4, there are two main kinds of software watermarking techniques: static and dynamic watermarks [23]. The nature of static watermarks makes it impracticable to transfer the embedded watermark to the container. As static watermarks are embedded in the executable file itself (i.e. they are not related to the program state), they cannot be used to build our dynamic containers. Hence, we need to use a dynamic watermark approach to generate the containers at runtime taking

into account the program state. Among the existing dynamic software watermarking approaches, the appropriate one for MAW is the "data structure watermark" because it is the one that takes into account the program state to generate the watermark (see section 2.4 and references [23, 19, 88]). In particular, in this design, it have been utilized the Collberg-Thomborson (CT) algorithm [23], which was summarized in Section 2.4.3.

Regarding the size of the MAW watermark, it is important because it determines the probability of detecting manipulation attacks. In the proposed scheme, the size of the marked code is determined before sending the agent. This size is limited and it is the same for all the hosts in the itinerary of the agent. Moreover, the container (output data) generated by each execution has also a limited size. The maximum size of containers is not arbitrary, it can be decided by the programmer to control the accuracy to detect manipulation attacks. If this proposal wants to improve detection ability and to prevent an attacker from modifying the code of the agent without being detected, it must increase the size of containers. Obviously, increasing the size of the watermark is also more costly in terms of transmission resources consumption. This is because the size of the agent is increased since on one hand, the marked code is bigger than the original one, and on the other hand, the containers are also bigger because they contain more redundancy. This last effect is even more significant since the agent carries a container for each execution in a host. This implies that the size of the agent grows as it traverses more hosts, and therefore this size depends on the length of the itinerary of the agent.

9.3.2 Watermark Transference

To detect manipulation attacks, the mobile agent must create at runtime the proofs, and send them back to the origin host to assure the execution correctness. In MAW, these proofs are stored in a logically-structured data "*container*" that is created in each host during execution. The container is created using dummy data, input data, internal values (heap, stack etc.) and data from external communications. The agent can diffuse and confuse all this information into the container to hide the actually desired execution results. Diffusing values means repeating these values into several different places, and confusing values means modifying these values to different ones, for example by adding constant values.

Obviously, all these data are not organized into the container at random. The way this information is incorporated into the container is essential to extract the watermark when the agent returns to the origin host. As it was said previously, the transferred watermark must be reliably located and extracted from the container and, it must let the agent owner demonstrate that its presence into the container is the result

of deliberate actions. In short, each executing host creates a container, which is the digital cover where the agent transfers the watermark. Then, each host must digitally sign its container. When the agent finishes traveling its itinerary, it returns to the origin host. All containers arrive at the origin host together with the mobile agent. Next, the origin host uses them to verify the execution integrity and to extract the desired execution results of each host.

9.3.3 Detecting Manipulations

The origin host must verify the execution integrity when the agent comes back with all the containers. To do so, the origin host uses its secret set of integrity rules related to the previously-embedded watermark. These integrity rules are a set of logical properties that a container must fulfill to demonstrate that it has not been tampered. They are also responsible for demonstrating that the presence of the watermark is the result of deliberate actions. These actions are inferred from the modifications performed over the original code of the agent code during the watermark embedding process. If a container does not fulfill the integrity rules, this means that the watermark has been modified, and the corresponding host is malicious. A tampered container can be used as a proof of the malicious behavior of a host. The host cannot repudiate this situation since it digitally signed the container.

It is worth to mention that the embedded watermark is the same for all the hosts, that is, all hosts execute the same marked code. In the same sense, the integrity rules are the same for all the hosts, because they are inferred directly from the code of the agent. This means that the origin host uses the same integrity rules to demonstrate the presence of the watermark into the containers. However, this does not mean that all the containers have the same data. Each container is different because it depends on the execution in each host, and hence the data used to fill in the container is different (input data, internal data, data from communications, dummy data, etc.). This could lead to think that this proposal uses fingerprinting instead of watermarking, because the data structure is different for each host (container). However, we consider that our approach uses watermarking because the embedded mark is the same for all the hosts despite the representation of this mark is different for each container.

Finally, our infrastructure also allows an origin host to prove, in front of an external third party, that a certain host of the itinerary performed a modification attack over the agent. However, the integrity rules cannot be treated as a proof directly. Instead, the origin host must send them together with the code of the agent and the signed container of the accused host to the third party. Then, the third party executes the agent several times with random input data. As any honest execution of the agent (independently of the input data) will generate valid containers, the new containers created

during these random executions should fulfill the integrity rules. This procedure assures that the integrity rules are valid. Then, the external entity can verify whether the host being accused is in fact malicious by applying the integrity rules to its container.

9.3.4 Advantages and Drawbacks of MAW

MAW is a lightweight attack detection approach if it is compared to the most widely known proposal of the cryptographic traces [112]. These are some of its advantages:

- **Size of the proofs:** in MAW, the size of the proofs to check the execution integrity is limited. The maximum size of the containers is determined by the programmer to control the accuracy to detect manipulation attacks. The containers can be little enough to let the agent carry them. In the cryptographic traces approach, the size of the traces depends on the amount of input data of the mobile agent, which can be quite big.

- **Proof storage:** in MAW, the executing hosts do not need to store any kind of proof. In the cryptographic traces approach, the hosts must store the traces for an indefinite period of time.

- **Hosts to verify:** in MAW, the origin host can verify the execution integrity of all the hosts of the itinerary. In the cryptographic traces approach the verification is performed in case of suspicion.

- **Verification tasks:** in MAW, the origin host has to apply the integrity rules to the containers to verify the execution integrity. In the cryptographic traces approach, the origin host must ask for the traces to the suspicious host and execute the agent again.

MAW has also some drawbacks, which affect mainly performance:

- **Watermark embedding:** the origin host must embed the watermark into the code of the agent by using software watermarking techniques and must infer the integrity rules.

- **Code size:** there is an increase in the code size. Embedding a watermark means that some overhead is added to the original code. This enlargement will depend on the embedded watermark and therefore, creating, storing and sending marked agents consume more resources.

- **Execution time:** the execution of marked agents consumes more CPU.

- **Mobile agent size:** the mobile agent in MAW must carry the containers. This implies an additional load. This load grows up each time the mobile agent visits a host. The maximum size is reached when the agent returns to the origin host.

9.3.5 Design of the Watermarks for MAW

This section discusses the motivations of malicious hosts, the attacks that can perform, and also the properties and requirements that the software watermarks should have to implement the MAW infrastructure.

9.3.5.1 Threat Model for MAW

As we mention in Section 2.4.2, the objective of an attacker in the copyright protection scenario is to make the watermark of a program invalid to illegally redistribute this program later. To do so, the main attacks against software watermarks are subtraction, addition and distortion. On the other hand, the motivations of an attacker in the mobile agent scenario are different. A malicious host may have several reasons to attack a mobile agent. For instance, it can attack the mobile agent to obtain some benefits from the execution, to damage the reputation of another host, or just for fun. There are several kinds of attacks in this scenario (denial of service, eavesdropping, impersonation, etc). However, we will focus on manipulation attacks because MAW has been designed to detect this particular kind of attacks. Just remind that manipulation attacks are those in which a malicious host tries to manipulate the proper execution of the agent to achieve a certain purpose. So then, the objective of an attacker will be not to make the watermark invalid, but to manipulate the execution without altering the transferred watermark, because any change in the transferred watermark will cause the detection of the attack.

The malicious host may try to manipulate the code of the agent to obtain a certain benefit. However, all the attacks that are used in the copyright protection scenario to manipulate code are totally useless to attack a mobile agent protected with the MAW infrastructure. Distortive attacks, which are usually based on semantic preserving program transformations (translation, optimization, obfuscation, etc.), are useless to attack MAW because these transformations only affect the code appearance, and not the code behavior. Hence, the modified code will be executed in the proper manner (which is precisely the objective, assuring the execution integrity). On the other hand, if a malicious host tries to remove the embedded watermark or to add a new one to the agent's code, the changes in the transferred watermark produced by these attacks will reveal that the agent has been modified.

The host can also try to attack containers. However, a host cannot manipulate the containers of previous executed hosts because they are signed by their creators. Thus,

a malicious host can only try to manipulate it own container. In this case, the objective of the attacker is manipulating the container to obtain a certain profit, but without altering the transferred watermark. However, this will be hard to achieve thanks to MAW because the host does not know which parts of the container are part of the watermark (this would be equivalent to know the integrity rules, which are secret). Therefore, before changing any part of the container the host should infer where the watermark is.

To infer where the watermark is, a malicious host can also try to analyze the inputs and outputs to extract information from the mobile agent. Unfortunately, it is unfeasible to detect or prevent that a host changes its own input data, which are located in its internal database. In fact, this should be considered an eavesdropping attack because the host does not alter the proper execution of the agent. As a conclusion, MAW cannot detect this because it is not a manipulation attack. However, MAW can avoid the attacker to extract information from the execution. Let us suppose that a malicious host introduces fake input data and executes the mobile agent to analyze the generated container. Despite this can be done many times obtaining different containers, this does not mean that the malicious host can generate a container at its discretion (containers must fulfill the integrity rules if the execution of the agent has not been modified). For this reason, a malicious host performing different executions cannot infer the integrity rules by comparing these containers because any change in the input data will cause that most data within the container will also change. In the same sense, colluding hosts that share their containers cannot infer where is the watermark. Even if a malicious host is successful obtaining some piece of information about how the containers are constructed, it would be unfeasible to construct a valid container that achieves the purposes of the attacker. This is due to the watermark being large and distributed within the whole container, and also because the attacker does not know all the integrity rules.

9.3.5.2 Watermark Properties

These are the most important properties of the watermarks to be embedded into the mobile agent:

1. the *stealth*. This is the most important property of the watermark, because a malicious host without knowledge about where the watermark is can only try random changes, which affect the transferred watermark;

2. the *data rate* is also quite important because it improves the security of the watermark. A bigger watermark makes manipulating the container without altering the transferred watermark more difficult. However, this affects adversely the

cost of the watermark, especially in terms of transmission resources as containers are sent back to the origin host;

3. the *resilience* is not as important as the previous properties, because the use of semantic preserving transformations does not affect the code behaviour. As a consequence, watermarks with little resilience can be used in the proposed scenario.

So, this proposal does not require maximizing all the properties. This allows us to use simpler and less costly watermarks.

9.3.6 Implementation of MAW using the CT Algorithm

The rest of this section introduces a brief description about how the MAW infraestructure have been designed. The Collberg-Thomborson (CT) algorithm [23] has been adapted to the mobile agent scenario. Again, it has been remarked that other different software watermarking algorithms could also be used to detect manipulation attacks. Obviously, the different peculiarities of each algorithm must be taken into account.

As it is explained in 2.4.3, the CT algorithm builds dynamically a graph in memory when the program is fed with a special input. The recognizer program is able to find this graph in memory, and to extract from this graph (for instance using the Radix-k encoding) a number N, which is the product of two large primes. As it is computationally unfeasible to factor a number which is the product of two large prime numbers, the creator demonstrates authorship by simply publishing the two factors.

In the design of MAW, all the executions build this graph in memory, independently on the input data that feeds the agent. So, the agent just needs to transfer this graph in memory to the container. The recognizer can also find N using the container as its input, instead of memory. In fact, the recognizer should be considered part of the integrity rules, which are more general as they also describe some more relationships among data within the container.

For simplicity, the proposed design is explained by means of a simple example. Suppose that we are executing the agent in the host n. The agent is fed with some input data that come from the previous execution host $n-1$. Also suppose that the initial state of the agent in this host has six values of this kind $s^n = (s_0^n, \cdots, s_5^n)$. After the execution, the agent will obtain some output data. In this example, suppose that the output have five of these values $o^n = (o_0^n, \cdots, o_4^n)$, which should be included within the container together with the transferred watermark.

The following steps have to be followed to construct the container:

1. The agent must calculate a binary initial sequence IS that will be used to establish the starting position of the watermark, and also to obscure the container.

This initial sequence IS should reflect that this execution has been performed in a certain host, that depends on the initial state, and that it is time dependent. In this example, IS is calculated as the hash of the concatenation of the identifier of host n (ID_n), a subset of the initial state (some values in clear and some hashed), and a timestamp TS_n:

$$IS = hash(ID_n||s_2^n||s_3^n||hash(s_5^n)||TS_n) \tag{9.1}$$

This time stamp TS_n should also be sent to the origin host together with the container to make possible to re-calculate IS from these values.

2. After that, the agent fills out the cells of the container with random values. Figure 9.1a shows the container at that moment. c_j represents the random data stored in the j position of the container.

C_0	C_1	C_2	C_3	C_4
C_5	C_6	C_7	C_8	C_9
C_{10}	C_{11}	C_{12}	C_{13}	C_{14}
C_{15}	C_{16}	C_{17}	C_{18}	C_{19}
C_{20}	C_{21}	C_{22}	C_{23}	C_{24}

(a) Container with random data

C_0	5	16	C_3	o^i_1
C_5	o^i_3	10	11	C_9
C_{10}	13	19	C_{13}	o^i_5
C_{15}	17	7	o^i_2	16
1	C_{21}	C_{22}	o^i_4	C_{24}

(b) Container after embedding the output data

a^i_3	5	16	a^i_2	o^i_1
a^i_4	o^i_3	10	11	a^i_1
C_{10}	13	19	a^i_5	o^i_5
C_{15}	17	7	o^i_2	16
1	C_{21}	C_{22}	o^i_4	C_{24}

(c) Container before XOR

Figure 9.1: Container generation process.

3. In this step, the agent starts transferring the watermark to the container. The first thing is locating the initial cell, p_n, of the graph into the container. In this example, the agent calculates p_n by hashing the initial sequence IS modulus 25 (25 is the number of cells of the container):

$$p_n = hash(IS)\ mod\ 25 = 7. \tag{9.2}$$

In this case, the initial sequence IS is the extra pointer which provides the position within the container of the first node within the circular linked graph (in this case, cell 7).

4. Next, the agent start transferring the rest of the graph from memory to the container. In this example, the watermark is the same of equation 2.14, which codifies the number $N = 4453$ with these Radix-6 coefficients

$$a = \{a_0, a_1, a_2, a_3, a_4\} = \{1, 4, 3, 2, 3\} \tag{9.3}$$

We denote the graph with the following set of tuples:

$$w^n = < \{x_0^n, y_0^n\}, \{x_1^n, y_1^n\}, \{x_2^n, y_2^n\}, \{x_3^n, y_3^n\}, \{x_4^n, y_4^n\} > \tag{9.4}$$

As in Figure 2.7, each node is formed by two elements (each of these element is mapped s in one cell, so it will need two consecutive cells for each graph node). The first cell is used to store the x_i^n value in base k, and it will be used to obtain the a_i Radix-6 coefficients. In this example, as $IS = 7$, cell 7 is the first cell of the first node of the graph, and hence it contains x_0^n. The second cell stores y_i^n, which is the index of the cell that corresponds to the next node of the graph. So then, cell 8 contains $y_0^n = 11$, which is a pointer to the second node of the graph (located in cells 11 and 12). Except the first node (that depends on IS), the agent can put the rest of nodes randomly in any place of the container, because we can reconstruct the complete graph using the pointers. Finally, the last node points to the first one (it is a circular linked graph). To difficult the location of the mark, the agent does not store the values directly into the container, but it calculates them as the subtraction of the value of the cell and the index of the cell $a_i = x_j^n - j$ (being j the index of this cell). For instance, $a_0 = 8 - 7 = 1$, $a_1 = 15 - 11 = 4$, $a_2 = 22 - 19 = 3$, and so on.

5. After the graph embedding, the agent must store the result of the execution $o^n = (o_0^n, \cdots, o_4^n)$. In this example, the agent chooses the positions to store these output values into the container using, another time, the value p_n (7, in our case). Basically, this value will indicate the number of empty cells between two different output values. The last node of the graph (in our case, cell 18) will be used as the starting point. So then, cell 4 stores the first output value o_1^n because it is 8th empty cell starting from cell 18. Figure 9.1b shows the container after transferring these values.

6. The following step is to generate an additional vector that will enhance the diffusion of the output data by storing them in some extra positions. In this case, we use a vector of 5 positions $o'^n = (o'^n_0, \cdots, o'^n_4)$, in which $o'^n_i = (o_i^n)^j \ mod \ p_n$. These values will be stored using the same rule than the o_i^n values (7 empty cells between different values), and starting from the last value added to the container (in this case o_5^n). Figure 9.1c shows the container after embedding the o'^n values. These relationships will also be part of the integrity rules.

7. Finally, all the container is XOR with IS to obscure all these data, so only the entities that know IS can know the real contents of the container.

To detect manipulations, the origin host should have the container, the initial state values $s^n = (s_0^n, \cdots, s_5^n)$ (sent by the previous host and acknowledged by host n), and the timestamp TS_n. Having all these data, the origin host can calculate IS. With that value, the origin host XOR the container using IS to recover the data within the container. Knowing IS, the initial cell of the watermark is also known. If host n has been honest, then $IS = 7$, and applying the recognizer to the container we can obtain that the Radix-6 values are $a = \{1,4,3,2,3\}$, so $N = 4453$. Just emphasize that N is not a random number but the product of two primes, and that the only entity capable of factoring it is the origin host ($N = 61 \times 73$). The origin host also verifies that the output values o^n and o'^n are properly located into the container and that the relationships among them are correct. If so, the infrastructure can consider that host n is honest, and the results can be extracted from the output values.

The security of this example lies on the impossibility of knowing which kind of data is storing each cell (watermark, output data, random data, etc). The executing host does not know how the initial sequence IS has been calculated, nor how the container has been constructed, nor the relationships among cells.

As it has been shown, data within the container are very dynamic, and they change as it changes the executing host, the time, the initial state of the agent or any other type of input data. However, the simplicity of this particular example must be stressed, which should be considered mainly didactic. The mechanisms used are preliminary, but that can be easily generalized and made more sophisticated to use in other scenarios that require more robustness. For this reason there are some vulnerabilities in this example. For instance, notice that we have put some order in the way to construct the container (first introduce the watermark, after that introduce the results, and at the end XOR the container). However, in a real scenario all these steps should be mixed, so the attacker cannot infer how the container has been constructed. In addition, some other relationships among the output values could be added to enhance diffusion and robustness, o''^n, o'''^n (for instance using arithmetical operations among them). Also, the watermark has been constructed using a Radix-k encoding that needs pointers, so all the cells that compose the graph has values in the range $[0-25]$. This can help an attacker to locate the watermark. In a real scenario these values should be obscured for instance by using any kind of arithmetical operation. Finally, a malicious host that provide the same initial state, the same host identifier, and the same timestamp to the agent will always obtain the same IS, so the position of the watermark will always be the same. Obviously, in a real example IS should change depending on other different parameters, so the attacker cannot infer where is the starting point of the watermark.

9.4 Punishing Attacks with the HoRA

This section introduces a punishment mechanism based on a new entity: the Host Revocation Authority (HoRA). The HoRA must be considered a Trusted Third Party (TTP) in the mobile agent system. In this sense, the HoRA must be considered a TTP in the mobile agent system like the Certification Authority (CA) is considered in the Public Key infrastructure (PKI). It seems logically that attack detection approaches should be accompanied with some punishment policies. Little attention has been paid to punishment mechanisms in mobile agent systems. In fact, this proposal is the only punishment system that can be found in the literature.

The HoRA uses a punishment mechanism based on host revocation. The aim of host revocation is to distinguish the malicious hosts from the honest ones. For this purpose, the HoRA stores a database with all the information related to past attacks. The main job of the HoRA is providing this information to the origin hosts to avoid new attacks from malicious hosts. In this chapter, the tasks that the HoRA has to carry out (status checking and host revocation) are summarized.

9.4.1 Status Checking

Before sending the agent, the origin host must consult the revocation information in order to delete malicious hosts from the itinerary of the agent. Assuming that the HoRA works in a similar way as the Certification Authority regarding certificate revocation, there are two possible ways of consulting the status of the hosts: online or offline. The decision of which of these policies must be used depends on multiple factors, like the available transmission resources, the number of origin hosts that may launch requests, or the computational capacity of the entities.

9.4.1.1 Offline Status Checking

In offline status checking, a reasonable assumption is considering that an origin host may lose the connectivity to the HoRA. If this happens, the origin host will not have any revocation information available. The idea behind the offline system is to make accessible the revocation information available in a given moment using a black list: the Host Revocation List (HRL). An HRL is a list, which is signed by the HoRA and, that contains all the identifiers of the hosts that have been revoked. The origin hosts can download the HRL and store it for some time. Then, the HRL can be used to remove revoked hosts from itineraries before sending agents. To take into account new malicious hosts, the origin hosts have to update the HRL periodically. In this sense, the HRL works in a similar way as the traditional Certificate Revocation List (CRL) in the PKI [60]. The origin hosts can download the HRL directly from the HoRA, but this may

cause a bottleneck in the system. To solve this problem the HoRA can put the HRL in repositories[1]. The repositories must also update the HRL periodically. The offline status checking mechanism does not avoid attacks completely. A host that is detected as malicious can attack agents until it is introduced in the HRL and the origin hosts update their lists. The less time between updates of the HRL, the less attacks can be performed by the new malicious hosts. However, frequent updates affect adversely the network bandwidth.

9.4.1.2 Online Status Checking

In the online status checking policy, origin hosts request revocation information directly from the HoRA. To do so, the origin hosts use the Online Host Status Protocol (OHSP). When a request arrives, the HoRA consults its internal database and sends a signed response pointing out the state of each host. This mechanism works in a similar way as the Online Certificate Status Protocol (OCSP) used in the PKI [86]. There are several reasons that can lead an origin host to use the online mechanism. For instance, origin hosts that send agents sporadically do not need to store and update the HRL periodically. Furthermore, the risk of suffering attacks from malicious hosts is minimized since the status is checked online which allows immediate detection and rejection of malicious hosts from itineraries. However, with the online policy, the HoRA may become a bottleneck in the system because it receives requests from all the origin hosts, and it must answer each request with a digitally signed response which is computationally expensive. For this reason, the HoRA can also delegate online checking to authorized entities called responders[2].

9.4.2 Host Revocation

The second task of the HoRA is managing the revocation information. As the revoked hosts are not removed from the database, this task consists mainly in adding new hosts. If the origin host has detected malicious hosts using MAW, it starts a protocol to revoke them. The objective of any revocation protocol is delivering in a reliable way all the proofs to the HoRA in order to demonstrate that an executing host is malicious. A proof is a piece of evidence that a TTP can use to verify that an attack was performed by a malicious host. In the case of MAW, the proofs are the containers, and the way to detect manipulation attacks are the set of integrity rules. Hence, the HoRA can only revoke a host in case there are proofs of its malicious behavior, that is, the HoRA needs evidences that demonstrate unmistakably that this host was malicious.

[1] A repository is a non-trusted location in the network where it is possible to store contents to make them available to download.

[2] A responder is a trusted location in the network that can send signed responses.

In this sense, the containers can be used to detect manipulation attacks because they have been signed by the hosts, so a malicious host cannot repudiate that it generated a certain container. The HoRA must verify that the set of integrity rules matches the code of the agent. This can be done by executing the agent several times with random input data (as explained in Section 9.3.3). A revocation protocol for MAW can be found in [35].

9.4.3 Summarizing the Overall Process

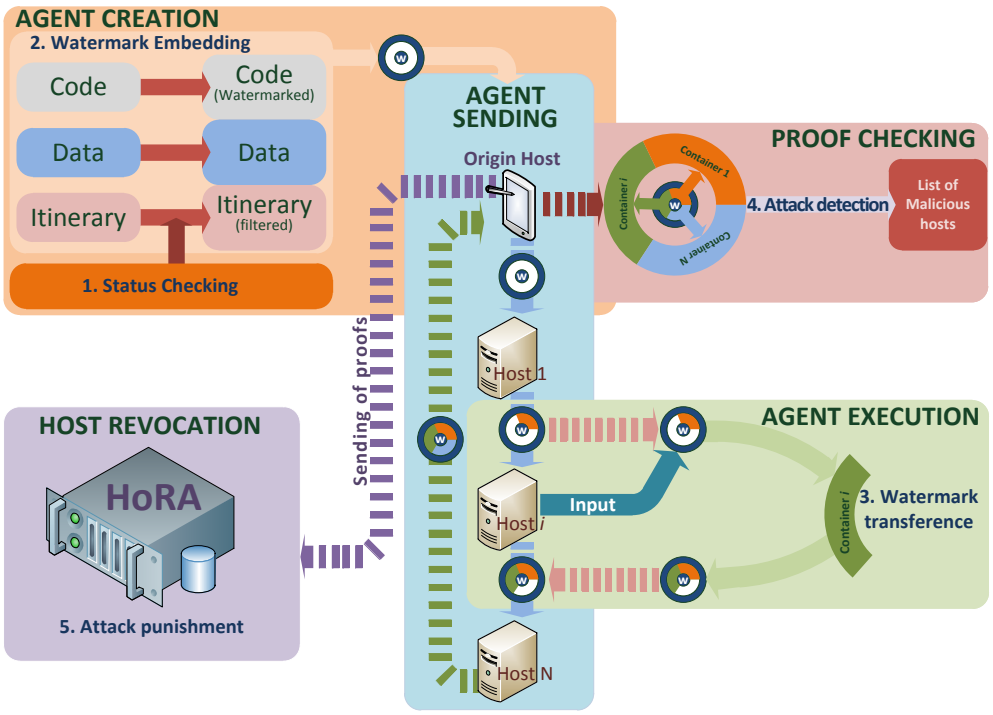


Figure 9.2: Working of MAW proposal

The lifetime of the agent can be divided in four phases (see Figure 9.2):

- **Agent Creation:** in this phase the origin host prepares the agent before sending it. This includes performing the status checking to filter the malicious hosts from the itinerary of the agent, and also embedding the watermark into the its code.

- **Agent Sending:** in this phase the origin host sends the mobile agent to perform its tasks. The agent will migrate from host to host executing its code and performing the actions that the user has programmed (for instance, arranging a meeting). During the agent's execution in each host, the agent must create and

store the containers, which are the proofs that will assure the execution integrity. The embedded watermark has been transferred to these containers during the execution. Hence, all the containers (one for each host) will return with the mobile agent to the origin host.

- **Proof Checking:** in this phase the origin host looks for malicious hosts. When the agent returns to the origin host, it extracts the containers of all the hosts and it verifies the signatures to detect possible errors in communications. These errors must not be considered manipulation attacks because they could be produced by the communication channel. If the containers do not have communication errors, the origin host applies the integrity rules to them in order to verify the correctness of the transferred watermarks. If a container does not fulfill the rules, this host is malicious and hence the origin host can start a revocation protocol.

- **Host Revocation:** in this phase the origin host sends the proofs of the execution integrity to the HoRA using a revocation protocol. The HoRA verifies the validity of these proofs, that is, it verifies that the signature of the container is valid and that the integrity rules correspond to the code of the agent by executing the agent again with random input data. If these proofs are valid, the HoRA revokes the malicious host, so this host will not receive agents any more.

9.5 Conclusions

In this chapter, the integration of two techniques are introduced to achieve an effective and usable protection mechanism for mobile agents against manipulation attacks performed by a malicious host during execution. On one hand, MAW has been presented as an effective and lightweight attack detection mechanism. The main ideas behind MAW have been explained, and a discussion about which are the most appropriate software watermarks to protect mobile agents is performed. In this way, the guidelines to correctly implement MAW using the CT algorithm have been also introduced. On the other hand, the HoRA has been presented as a generic TTP with punishment capabilities. The combined use of the two security mechanisms leads to a reliable environment for honest users and hosts, which is worth even at the expense of introducing some overhead.

Part V

Final remarks

CHAPTER 10

CONCLUSIONS AND FUTURE WORK

This chapter summarizes the conclusions of this thesis (section 10.1) and outlines future lines of research (section 10.2).

10.1 Conclusions

The main contributions related to fingerprinting codes and schemes could be condensed in the following points:

- A new problem in Collusion Secure Convolutional Fingerprinting Information Codes that generates false positives has been shown. As a result of analysing the work by Zhu *et* al. in [122], the drawbacks of not considering false positive have been enlightened. The original results in [122] have been revisited from the point of view of the false positive problem. Moreover the probability of false positives has been quantified formally and contrasted with simulations. Furthermore some guidelines for a correct design of Collusion Secure Convolutional Fingerprinting Information Codes are given.

- After analysing the turbo fingerprinting codes presented by Zhang *et* al. in [121], we showed that the probability of tracing one of the traitors tends to 0 when the alphabet size of the outer turbo code increases. This is because the symbol-by-symbol collusion attack performed by pirates is not treated efficiently by the decoding algorithm proposed in [121]. Note that, from the point of view of the turbo decoder, the error probability of the equivalent channel tends to $1/2$, because it takes as input symbols one of the symbols retrieved by the Boneh-Shaw

decoder chosen at random. The new problem found in the turbo fingerprinting codes renders them inapplicable in many cases unless the design takes into account our new contribution. Moreover, our studies indicate that, the more efficient the turbo fingerprinting code design is, from the point of view of the length requirement, a far worse performance is obtained from the tracing algorithm. In other words, to find a traitor will be more complicated when the (n,k)-turbo code used has large values of n. Besides, two different ways to improve the performance of turbo fingerprinting codes are given. These two ways use the likelihood of the turbo decoder to perform the improvements. The first proposal modifies this likelihood at the input of the turbo decoder and the other uses the turbo decoder output likelihood to correlate it with the user IDs in order to find the traitors. Moreover, these two improvements can be integrated in the same scheme.

- Turbo fingerprinting codes based on MFD low-rate convolutional codes have been proposed as a family of fingerprinting codes secure against coalitions of size 2. It is shown by simulation that the proposed codes identify traitors with an error probability of at most $5 * 10^{-4}$ when the number of users lies between 2^8 and 2^{32}. Moreover, when the number of colluders is $c = 3$, our system performance is also acceptable with a $P_e < 0.01$ even when the noise added by the colluders degrades the channel to an SNR of 4dB. Finally, it is important to stress that the presented codes have an efficient decoding algorithm based on the Max-Log-MAP iterative decoding algorithm.

The contributions related to secure e-commerce of multimedia content, could be put in a nutshell as:

- We have developed a proof of concept that tracing traitors over YouTube video service is possible. First of all, the relation between bit error probability, watermarking robustness and distortion is deeply studied. Our study shows that a nice trade-off between these parameters can be achieved. Next, our conclusions are applied to the problem of traitor tracing. In this way, we use the watermarking layer (configured taking into account our results) with a fingerprinting code. It is shown that this fingerprinting code can trace traitors if no collusion is performed, with a really low distortion. If collusion appears, the traitors could be also traced with a low distortion. Besides, the proposed system does not allow false positives by design, that is, innocent users cannot be framed.

- A platform for the delivery of copyright protected digital content has been implemented. The copyright protection is based on a combination of watermarking and fingerprinting techniques that allow the generation of protected copies

that can be distributed in a trusted environment. The dishonest redistribution of the purchased contents can be traced by means of watermarking extraction and fingerprinting analysis. Both techniques result in the identification of the malicious users involved in the unauthorized redistribution process. The solution is presented as a distributed JEE application based on standardized frameworks such as Spring and Hibernate. As a result the system offers loosely coupled Web services with a high cohesion. The independence and modularity of the services invite for good maintainability and high scalability. The platform presents a demonstrable evidence of a copyright protection system that proves the feasibility of the implementation of watermarking and fingerprinting algorithms on a real life scenario.

Finally, in relation to mobile agents protection, the following contributions have been done:

- Attacks from malicious host against agents are considered one of the most difficult problems to solve and there is not a way to avoid them. Furthermore, the use of mobile agents in IDS is increasingly common so, in order to offer the required security level in IDS based on agents, it is necessary to combine different techniques to detect a possible attack although it can not be avoided. The drawback of sending an agent to a host is that the host could be malicious and it can attack the agent because it has whole access not only to the agent data but the code. Two different alternatives based on the use of software watermarking techniques are presented. The first one is the use of 2 entities for each network segment which control the activity of this network. After a fixed period of time, one of these entities is moved on to the monitor in order to be verified. The second proposal uses a cooperative itinerant agent to verify into the monitored host the agent in charge of a network segment. This system uses a matrix of marks and determines if an agent was modified by asking subsequently set of sub-marks. Moreover, in these proposals, tamper-proofing and obfuscation techniques are used to make harder the work of modifying an agent execution.

- A novel approach to guarantee the security of the mobile agents execution in a compromised host is presented. This approach is based on the algorithm presented by Myles *e*t al. in [87]. This algorithm has, as well as authorship protection capacities, integrity code check, tamper detection and fingerprinting capacities. The original algorithm is based upon branch functions that transfer the agent execution to the correct point in the program execution flow from a calculation obtained of one way function which depends on an authorship mark and an integrity check. Our improvement is the incorporation of an external el-

ement called **sentinel** which keeps the relation or mapping between the values obtained from the one way function (k_i) and the pointer to the next instruction in the program execution flow. The controlled host needs to send the value of the appropriate k_i to the **sentinel** and waiting for the response to execute the agent in the correct way. On the other hand, the **sentinel** can control periodically the agent execution because it knows the correct sequence of k_i for each agent. Additionally, a security analysis of the proposed schema is presented and some aspects related to its implementation are commented.

- The integration of two techniques are introduced to achieve an effective and usable protection mechanism for mobile agents against manipulation attacks performed by a malicious host during execution. On one hand, MAW has been presented as an effective and lightweight attack detection mechanism. The main ideas behind MAW have been explained, and a discussion about which are the most appropriate software watermarks to protect mobile agents is performed. In this way, the guidelines to correctly implement MAW using the CT algorithm have been also introduced. On the other hand, the HoRA has been presented as a generic TTP with punishment capabilities. The combined use of the two security mechanisms leads to a reliable environment for honest users and hosts, which is worth even at the expense of introducing some overhead.

10.2 Future research work

As future research work, we present a list of open problems

- It would be interesting to study a list-decoding algorithm suitable for turbo-decoding to obtain the two most likely colluders.
- A basic element in the turbo-codes is the interleaver. In this thesis we have focused on the constituent codes. It would be interesting to study what should be the best interleaver for a turbo-fingerprinting code.
- During this thesis we have seen that the convolucional codes can be modified in order to be used in fingerprinting. On the other hand it has been seen that the concatenation of algebraic codes can give a good result in this area. A possible future line could be to study how a concatenation of convolutional codes with algebraic codes would be adapted to the problem of tracing traitors.
- With regard to the integration of watermarking and fingerprinting it seems interesting to study how to scale the marking process in order to be able to distribute large amounts of video in real time correctly marked with the user information of the buyer and thus prevent redistribution.

- The marking software developed in this thesis focuses on the MPEG-2 format. Although we have tried to recover the mark despite various format changes, it would be interesting to see how to take advantage of the properties of the new video formats to embed marks.
- It would be interesting to see how to bring all the logic of protection and reputation of mobile agents to the Internet of Things. This new paradigm has similarities with mobile agents since we have a number of code elements that run on networks or devices that may have an interest in manipulating the execution of this code.

OWN REFERENCES

JCR

[EMTBS11] O. Esparza, J. L. Muñoz, J. Tomàs-Buliart, and M. Soriano. An infrastructure for detecting and punishing malicious hosts using mobile agent watermarking. *Wireless Communications and Mobile Computing*, 11:1446 – 1462, 11 2011.

[TBFS09] Joan Tomàs-Buliart, Marcel Fernández, and Miguel Soriano. Traitor tracing over youtube video service - proof of concept. *Telecommunication Systems*, 45(1):47–60, 2009.

LNCS

[PTBFS07] Rafael Pàez, Joan Tomàs-Buliart, Jordi Forné, and Miguel Soriano. Securing agents against malicious host in an intrusion detection system. In *Critical Information Infrastructures Security (CRITIS 2007)*, volume 5141/2008 of *Lecture Notes in Computer Science*, pages 94–105. Springer Berlin / Heidelberg, 2007.

[SFS+05] Miguel Soriano, Marcel Fernandez, Elisa Sayrol, Joan Tomas, Joan Casanellas, Josep Pegueroles, and Juan Hernández-Serrano. Multimedia copyright protection platform demonstrator. In *Trust Management (iTrust '05)*, volume 3477/2005 of *Lecture Notes in Computer Science*, pages 159–178. Springer Berlin / Heidelberg, 2005.

[TBFS07] Joan Tomàs-Buliart, Marcel Fernandez, and Miguel Soriano. Using informed coding and informed embedding to design robust fingerprinting embedding systems. In *Knowledge-Based Intelligent Information and Engineering Systems(KES 2007)*, volume 4694/2007 of *Lecture Notes in Computer Science*, pages 992–999. Springer Berlin / Heidelberg, 2007.

[TBFS08a] J. Tomàs-Buliart, M. Fernández, and M. Soriano. New considerations about the correct design of turbo fingerprinting codes. In *Computer Se-*

curity - ESORICS 2008, volume 5283/2008 of *Lecture Notes in Computer Science*, pages 501–516. Springer Berlin / Heidelberg, 2008.

[TBFS08b] J. Tomàs-Buliart, M. Fernández, and M. Soriano. Protection of mobile agents execution using a modified self- validating branch-based software watermarking with external sentinel. In *Critical Information Infrastructure Security (CRITIS 2008)*, volume 5508/2009 of *Lecture Notes in Computer Science*, pages 287–294. Springer Berlin / Heidelberg, 2008.

[TBFS09] Joan Tomàs-Buliart, Marcel Fernandez, and Miguel Soriano. Improvement of collusion secure convolutional fingerprinting information codes. In *Information Theoretic Security (ICITS' 07)*, volume 4883/2009 of *Lecture Notes in Computer Science*, pages 76–88. Springer Berlin / Heidelberg, 2009.

International conferences

[PTBFS07] Rafael Pàez, Joan Tomàs-Buliart, Jordi Forné, and Miguel Soriano. Mais: Mobile agent integrity system. a security system to ids based on autonomous agents. In *Proceedings of the International Conference on Security and Cryptography (SECRYPT'07)*, 2007.

[SFT+05] Miguel Soriano, Stephan Flake, Juergen Tacken, Frank Borman, and Joan Tomàs-Buliart. Digital rights management specification for nomadic services. In *Proceedings of the 16th International Workshop on Database and Expert Systems Applications (DEXA 2005)*, 2005.

[SSF+05] Elisa Sayrol, Miguel Soriano, Marcel Fernandez, Joan Casanellas, and Joan Tomas. Development of a platform offering video copyright protection and security against illegal distribution. In *Security, Steganography, and Watermarking of Multimedia Contents*, pages 76–83, 2005.

[TBGMFS11] J. Tomas-Buliart, A. Gomez-Muro, M. Fernandez, and M. Soriano. Use of turbo codes with low-rate convolutional constituent codes in fingerprinting scenarios. In *Information Forensics and Security (WIFS), 2011 IEEE International Workshop on*, pages 1–6, 2011.

Spanish conferences

[TBCFS06] Joan Tomàs-Buliart, Marc Ciurana, Marcel Fernández, and Miguel Soriano. Watermarking de software: Estado del arte. In *IX Reunión Española sobre Criptología y Seguridad de la Información*, 2006.

[TBCSF05] Joan Tomàs-Buliart, Joan Casanellas, Miguel Soriano, and Marcel Fernández. Diseño de una plataforma de protección del copyright de vídeo. In *III Simposio Español de Comercio Electrónico (SCE'05)*, 2005.

[TBdCSP07] Joan Tomàs-Buliart, Juan Vera del Campo, Miguel Soriano, and Josep Pegueroles. Diseño seguro de una plataforma de e-gobierno. In *VI Jornadas de Ingeniería Telemática*, 2007.

[TBSF05] Joan Tomàs-Buliart, , Miguel Soriano, and Marcel Fernández. Integrando watermarking y fingerprinting para protección de mpeg-2. In *Telecom i+d 2005*, 2005.

[VTBFS10] Sergi Vendrell, Joan Tomàs-Buliart, Marcel Fernandez, and Miguel Soriano. Estudio sobre el uso de códigos ldpc en esquemas de fingerprinting. In *XI Reunión Española sobre Criptología y Seguridad de la Información*, 2010.

REFERENCES

[1] R.J. Anderson and F.A.P. Petitcolas. On the limits of steganography. *Selected Areas in Communications, IEEE Journal on*, 16(4):474–481, May 1998.

[2] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv. Optimal decoding of linear codes for minimizing symbol error rate (corresp.). *Information Theory, IEEE Transactions on*, 20(2):284–287, Mar 1974.

[3] J. S. Balasubramaniyan, Garcia J. O. Fernandez, D. Isacoff, Eugene H. Spafford, and Diego Zamboni. An architecture for intrusion detection using autonomous agents. In *ACSAC*, pages 13–24, 1998.

[4] S. Baluja and M. Covell. Audio fingerprinting: Combining computer vision & data stream processing. In *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing ICASSP 2007*, volume 2, pages II–213–II–216, 15–20 April 2007.

[5] S. Baluja, M. Covell, and S. Ioffe. Permutation grouping: intelligent hash function design for audio &image retrieval. In *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing ICASSP 2008*, pages 2137–2140, March 31 2008–April 4 2008.

[6] Alexander Barg, G. R. Blakley, and Gregory A. Kabatiansky. Digital fingerprinting codes: problem statements, constructions, identification of traitors. *IEEE Transactions on Information Theory*, 49(4):852–865, 2003.

[7] Tom Bellwood, David Ehnebuske, Yin Leng Husband, Alan Karp, Keisuke Kibakura, Jeff Lancelle, Sam Lee, Sean MacRoibeaird, Barbara McKee, Tammy Nordan, Dan Rogers, Christine Tomlinson, and Cafer Tosun. Uddi version 2.03 data structure reference, July 2002.

[8] W. R. Bender, D. Gruhl, N. Morimoto, and A. Lu. Techniques for data hiding. In W. Niblack and R. C. Jain, editors, *Proc. SPIE Vol. 2420, p. 164-173, Storage and Retrieval for Image and Video Databases III, Wayne Niblack; Ramesh C. Jain;*

Eds., volume 2420 of *Presented at the Society of Photo-Optical Instrumentation Engineers (SPIE) Conference*, pages 164–173, March 1995.

[9] C. Berrou and A. Glavieux. Near optimum error correcting coding and decoding: turbo-codes. *Communications, IEEE Transactions on*, 44(10):1261–1271, Oct 1996.

[10] C. Berrou, A. Glavieux, and P. Thitimajshima. Near shannon limit error-correcting coding and decoding: Turbo-codes. 1. *Communications, 1993. ICC 93. Geneva. Technical Program, Conference Record, IEEE International Conference on*, 2:1064–1070 vol.2, 23-26 May 1993.

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

[12] Dan Boneh and James Shaw. Collusion-secure fingerprinting for digital data (extended abstract). In *CRYPTO '95: Proceedings of the 15th Annual International Cryptology Conference on Advances in Cryptology*, pages 452–465, London, UK, 1995. Springer-Verlag.

[13] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, and Dave Winer. Simple object access protocol (soap) 1.1. Technical report, May 2000.

[14] D. Chase. A class of algorithms for decoding block codes with channel measurement information. *Information Theory, IEEE Transactions on*, 18(1):170–182, Jan 1972.

[15] Brian Chen, Gregory W. Wornell, and Senior Member. Quantization index modulation: A class of provably good methods for digital watermarking and information embedding. *IEEE Trans. on Information Theory*, 47:1423–1443, 2001.

[16] D. Chess. Security considerations in agent-based systems. In *First IEEE Conference on Emerging Technologies and Applications in Communications (etaCOM)*, 1996.

[17] D. Chess. Security issues in mobile code systems. In *Mobile Agents and Security*, volume 1419 of *LNCS*. Springer-Verlag, 1998.

[18] Benny Chor, Amos Fiat, Moni Naor, and Benny Pinkas. Tracing traitors. *IEEE Transactions on Information Theory*, 46(3):893–910, 2000.

[19] C. Collberg, E. Carter, S. Debray, A. Huntwork, J. Kececioglu, C. Linn, and M. Stepp. Dynamic path-based software watermarking. *SIGPLAN Not.*, 39(6):107–118, 2004.

[20] C. Collberg, G. Myles, and A. Huntwork. Sandmark - a tool for software protection research. *IEEE Security and Privacy*, 1(4), 2003.

[21] C. Collberg and C. Thomborson. Watermarking, tamper-proofing, and obfuscation - tools for software protection. *IEEE Transactions on Software Engineering*, 28(8):735–746, 2002.

[22] Christian Collberg and Clark Thomborson. On the limits of software watermarking. Technical Report 164, Department of Computer Science, The University of Auckland, August 1998.

[23] Christian Collberg and Clark Thomborson. Software watermarking: Models and dynamic embeddings. In *Principles of Programming Languages 1999, POPL'99*, San Antonio, TX, January 1999.

[24] Max H. M. Costa. Writing on dirty paper. *IEEE Transactions on Information Theory*, 29(3):439–, 1983.

[25] J. Cotrina-Navau and M. Fernandez. A family of asymptotically good binary fingerprinting codes. *IEEE Trans. Inform. Theory*, 56(10), 2010.

[26] M. Covell and S. Baluja. Known-audio detection using waveprint: Spectrogram fingerprinting by wavelet hashing. In *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing ICASSP 2007*, volume 1, pages I–237–I–240, 15–20 April 2007.

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

[28] Ingemar Cox, Matthew Miller, Jeffrey Bloom, Jessica Fridrich, and Ton Kalker. *Digital Watermarking and Steganography*. Morgan Kaufmann, 2007.

[29] Ingemar Cox, Matthew L. Miller, and Jeffery A. Bloom. *Digital watermarking*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[30] Dorothy E. Denning. An intrusion-detection model. *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, 13(2):222–232, 1987.

[31] J. Domingo-Ferrer and J. Herrera-Joancomarti. Simple collusion-secure fingerprinting schemes for images. *Information Technology: Coding and Computing, 2000. Proceedings. International Conference on*, pages 128–132, 2000.

[32] Greg Meredith Erik Christensen, Francisco Curbera and Sanjiva Weerawarana. Web services description language (wsdl) 1.1, March 2001.

[33] O. Esparza, J.L. Muñoz, M. Soriano, and J. Forné. Punishing malicious hosts with the cryptographic traces approach. *New Generation Computing*, 24(4):351–376, 2006.

[34] O. Esparza, M. Soriano, J.L. Munoz, and J. Forne. A protocol for detecting malicious hosts based on limiting the execution time of mobile agents. In *Computers and Communication, 2003. (ISCC 2003). Proceedings. Eighth IEEE International Symposium on*, pages 251–256 vol.1, 2003.

[35] O. Esparza, M. Soriano, J.L. Munoz, and J. Forne. Punishing manipulation attacks in mobile agent systems. In *Global Telecommunications Conference, 2004. GLOBECOM '04. IEEE*, volume 4, pages 2235–2239 Vol.4, 2004.

[36] William M. Farmer, Joshua D. Guttman, and Vipin Swarup. Security for mobile agents: Issues and requirements. In *In Proceedings of the 19th National Information Systems Security Conference*, pages 591–597, 1996.

[37] W.M. Farmer, J.D. Guttmann, and V. Swarup. Security for mobile agents: Authentication and state appraisal. In *European Symposium on Research in Computer Security (ESORICS)*, volume 1146 of *LNCS*. Springer-Verlag, 1996.

[38] M. Fernandez and Miguel Soriano. Fingerprinting concatenated codes with efficient identification. In *ISC '02: Proceedings of the 5th International Conference on Information Security*, pages 459–470, London, UK, 2002. Springer-Verlag.

[39] Jr. Forney, G. Convolutional codes i: Algebraic structure. *Information Theory, IEEE Transactions on*, 16(6):720–738, 1970.

[40] Stan Franklin and Art Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *ECAI '96: Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages*, pages 21–35, London, UK, 1997. Springer-Verlag.

[41] Elke Franz, Anja Jerichow, Steffen Möller, Andreas Pfitzmann, and Ingo Stierand. Computer based steganography: How it works and why therefore any restrictions on cryptography are nonsense, at best. *Information Hiding*, pages 7–21, 1996.

[42] Pål Frenger, Pål Orten, Pal Frenger, Pal Orten, and Tony Ottosson. Code-spread cdma using maximum free distance low-rate convolutional codes. *IEEE Transactions on Communications*, 48:135–144, 2000.

[43] Jiri Fridrich. A new steganographic method for palette-based images. In *PICS 1999: Proceedings of the Conference on Image Processing, Image Quality and Image Capture Systems (PICS-99)*, pages 285–289. IS&T - The Society for Imaging Science and Technology, 1999.

[44] Robert G. Gallager. *Low Density Parity-Check Codes*. PhD thesis, MIT, Cambridge, 1963.

[45] Ricardo Garcia Gonzalez. youtube-dl: Download videos from youtube.com. http://www.arrakis.es/ rggi3/youtube-dl/.

[46] Inc. Google. Youtube apis and tools. http://code.google.com/apis/youtube/overview.html.

[47] B. Goyal, S. Sitaraman, and S. Krishnamurthy. Intrusion detection system: An overview. SANS Institute 2001, as part of the Information Security Reading Room., 2003.

[48] K. Gracie and M.-H. Hamon. Turbo and turbo-like codes: Principles and applications in telecommunications. *Proceedings of the IEEE*, 95(6):1228–1254, 2007.

[49] Hans-Jürgen Guth and Birgit Pfitzmann. Error- and collusion-secure fingerprinting for digital data. In *IH '99: Proceedings of the Third International Workshop on Information Hiding*, pages 134–145, London, UK, 2000. Springer-Verlag.

[50] J. Hagenauer and P. Hoeher. A viterbi algorithm with soft-decision outputs and its applications. *Global Telecommunications Conference, 1989, and Exhibition. 'Communications Technology for the 1990s and Beyond'. GLOBECOM '89., IEEE*, pages 1680–1686 vol.3, 27-30 Nov 1989.

[51] J. Hagenauer, E. Offer, and L. Papke. Iterative decoding of binary block and convolutional codes. *Information Theory, IEEE Transactions on*, 42(2):429–445, 1996.

[52] J. Hagenauer and L. Papke. Decoding turbo-codes with the soft output viterbi algorithm (sova). *Information Theory, 1994. Proceedings., 1994 IEEE International Symposium on*, pages 164–, 27 Jun-1 Jul 1994.

[53] R. W. Hamming. Error detecting and error correcting codes. *Bell System Techincal Journal*, 29:147–160, 1950.

[54] Frank Harary and Edgar M. Palmer. *Graphical Enumeration*. Academic Press, 1973. Academic Press, New York.

[55] S He and M Wu. Performance study on multimedia fingerprinting employing traceability code. In *IEEE International Workshop on Digital Watermarking*, volume 3710, pages 84–96, Siena, Italy, September 2005.

[56] Shan He and Min Wu. Collusion-resistant video fingerprinting for large user group. *Image Processing, 2006 IEEE International Conference on*, pages 2301–2304, Oct. 2006.

[57] Shan He and Min Wu. Joint coding and embedding techniques for multimediafingerprinting. *Information Forensics and Security, IEEE Transactions on*, 1(2):231–247, June 2006.

[58] Fritz Hohl. Time limited blackbox security: Protecting mobile agents from malicious hosts. In *Mobile Agents and Security*, pages 92–113, London, UK, 1998. Springer-Verlag.

[59] Henk D. L. Hollmann, Jack H. van Lint, Jean-Paul Linnartz, and Ludo M. G. M. Tolhuizen. On codes with the identifiable parent property. *Journal of Combinatorial Theory, Series A*, 82(2):121–133, May 1998.

[60] R. Housley, W. Ford, W. Polk, and D. Solo. Internet x.509 public key infrastructure certificate and crl profile, 1999. RFC 2459.

[61] W. Jansen and T. Karygiannis. Mobile agent security. Special publication 800-19, National Institute of Standards and Technology (NIST), 1999.

[62] W. Jansen, P. Mell, T. Karygiannis, and D. Marks. Mobile agents in intrusion detection and response. In *Proc. 12th Annual Canadian Information Technology Security Symposium*, Ottawa, 2000.

[63] Wayne A. Jansen. Countermeasures for mobile agent security. *Computer Communications*, 23(17):1667–1676, November 2000.

[64] Neil F. Johnson and Sushil Jajodia. Steganalysis of images created using current steganography software. In *Proceedings of the Second International Workshop on Information Hiding*, pages 273–289, London, UK, 1998. Springer-Verlag.

[65] N.F. Johnson and S. Jajodia. Steganalysis: the investigation of hidden information. *Information Technology Conference, 1998. IEEE*, pages 113–116, 1-3 Sep 1998.

[66] Jean-François Jourdas and Pierre Moulin. High-rate random-like spherical fingerprinting codes with linear decoding complexity. *Trans. Info. For. Sec.*, 4:768–780, December 2009.

[67] Taekyung Kim, Taesuk Oh, and Yong Cheol Kim. Fast informed embedding in dirty-paper trellis-code with orthogonal arcs. In *MCPS '06: Proceedings of the 4th ACM international workshop on Contents protection and security*, pages 47–52, New York, NY, USA, 2006. ACM Press.

[68] D. Kinny. Reliable agent communication - a pragmatic perspective. *New Generation Computing*, 19(2):139–156, 2001.

[69] R. Koetter and A. Vardy. Algebraic soft-decision decoding of reed-solomon codes. *Information Theory, IEEE Transactions on*, 49(11):2809–2825, Nov. 2003.

[70] C. Kurak and J. McHugh. A cautionary note on image downgrading. *Computer Security Applications Conference, 1992. Proceedings., Eighth Annual*, pages 153–159, Dec 1992.

[71] Matthew Kwan. The gifshuffle home page.

[72] Danny B Lange and Mitsuru Oshima. *Programming and Deploying Java Mobile Agents with Aglets*. Addison-Wesley, Reading, MA, 1998.

[73] S. Lin and D. J. Costello, Jr. *Error Control Coding: Fundamentals and Applications*. Prentice-Hall, Englewood Cliffs, New Jersey, 1983.

[74] Shu Lin and Daniel J. Costello. *Error Control Coding, Second Edition*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2004.

[75] T. Lindkvist. Fingerprinting of digital documents. Dissertation, 2001.

[76] Cullen Linn and Saumya Debray. Obfuscation of executable code to improve resistance to static disassembly. In *CCS '03: Proceedings of the 10th ACM conference on Computer and communications security*, pages 290–299, New York, NY, USA, 2003. ACM Press.

[77] Romana Machado. Ez stego. http://www.fqa.com.

[78] Macromedia. Macromedia and sorenson media bring video to macromedia flash content and applications. Macromedia Press Room, 2002.

[79] A. Maña, J. Lopez, J.J. Ortega, E. Pimentel, and J.M. Troya. A framework for secure execution of software. *International Journal of Information Security*, 3(2):99–112, 2004.

[80] Matthew L. Miller, Gwenaël J. Doërr, and Ingemar J. Cox. Dirty-paper trellis codes for watermarking. In *ICIP (2)*, pages 129–132, 2002.

[81] Matthew L. Miller, Gwenaël J. Doërr, and Ingemar J. Cox. Informed embedding for multi-bit watermarks. In *Digital Watermarking: First International Workshop, IWDW 2002, Seoul, Korea, November 21-22, 2002. Revised Papers*, pages 13–21, November 2002.

[82] Matthew L. Miller, Gwenaël J. Doërr, and Ingemar J. Cox. Applying informed coding and embedding to design a robust high-capacity watermark. *IEEE Transactions on Image Processing*, 13(6):792–807, 2004.

[83] Y. Minsky, R. van Renesse, F. Schneider, and S.D. Stoller. Cryptographic support for fault-tolerant distributed computing. In *Seventh ACM SIGOPS European Workshop*, 1996.

[84] P. Moulin and R. Koetter. Data-hiding codes. *Proceedings of the IEEE*, 93(12):2083–2126, Dec. 2005.

[85] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. X.509 internet public key infrastructure - online certificate status protocol - ocsp. June 1999.

[86] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. X.509 internet public key infrastructure online certificate status protocol - ocsp, 1999. RFC 2560.

[87] Ginger Myles and Hongxia Jin. Self-validating branch-based software watermarking. In *Information Hiding*, pages 342–356, 2005.

[88] Jasvir Nagra and Clark D. Thomborson. Threading software watermarks. In *Information Hiding*, pages 208–223, 2004.

[89] Josep Cotrina Navau, Marcel Fernandez, and Miguel Soriano. A family of collusion 2-secure codes. In auro Barni, Jordi Herrera-Joancomartí, Stefan Katzenbeisser, and Fernando Pérez-González, editors, *Information Hiding, 7th International Workshop, IH 2005, Barcelona, Spain, June 6-8, 2005, Revised Selected Papers*, volume 3727 of *Lecture Notes in Computer Science*, pages 387–397. Springer, 2005.

[90] G. Necula and P. Lee. Safe, untrusted agents using proof-carrying code. In *Mobile Agents and Security*, volume 1419 of *LNCS*. Springer-Verlag, 1998.

[91] Koji Nuida. An improvement of short 2-secure fingerprint codes strongly avoiding false-positive. In Stefan Katzenbeisser and Ahmad-Reza Sadeghi, editors, *Information Hiding*, volume 5806 of *Lecture Notes in Computer Science*, pages 161–175. Springer Berlin / Heidelberg, 2009.

[92] Hyacinth S. Nwana. Software agents: An overview. *Knowledge Engineering Review*, 11:205–244, 1996.

[93] R. Oppliger. Security issues related to mobile code and agent-based systems. *Computer Communications*, 22(12):1165–1170, 1999.

[94] Rafael Paez, Cristina Satizabal, and Jordi Forne. Cooperative itinerant agents (cia): Security scheme for intrusion detection systems. In *ICISP '06: Proceedings of the International Conference on Internet Surveillance and Protection*, page 26, Washington, DC, USA, 2006. IEEE Computer Society.

[95] Mark A. Pinsky. *Introduction to Fourier Analysis and Wavelets (Brooks/Cole Series in Advanced Mathematics)*. Thomson Brooks/Cole, 2001.

[96] I. Pitas. A method for signature casting on digital images. *Image Processing, 1996. Proceedings., International Conference on*, 3:215–218 vol.3, Sep 1996.

[97] FFmpeg Project. Ffmpeg. http://ffmpeg.org/.

[98] Reihaneh Safavi-Naini and Yejing Wang. Collusion secure q-ary fingerprinting for perceptual content. In *DRM '01: Revised Papers from the ACM CCS-8 Workshop on Security and Privacy in Digital Rights Management*, pages 57–75, 2002.

[99] Tomas Sander and Christian F. Tschudin. Protecting mobile agents against malicious hosts. In *Mobile Agents and Security*, pages 44–60, London, UK, 1998. Springer-Verlag.

[100] A. De Santis and G. Persiano. Zero-knowledge proofs of knowledge without interaction. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:427–436, 1992.

[101] Bruce Schneier. *Applied cryptography (2nd ed.): protocols, algorithms, and source code in C*. John Wiley & Sons, Inc., New York, NY, USA, 1995.

[102] Francesc Sebé and Josep Domingo-Ferrer. Short 3-secure fingerprinting codes for copyright protection. *Information Security and Privacy*, pages 279–283, 2002.

[103] Claude Elwood Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 7 1948.

[104] Jessica Staddon, Douglas R. Stinson, and Ruizhong Wei. Combinatorial properties of frameproof and traceability codes. *IEEE Transactions on Information Theory*, 47(3):1042–1049, March 2001.

[105] D. R. Stinson, Tran van Trung, and R. Wei. Secure frameproof codes, key distribution patterns, group testing algorithms and related structures. *Journal of Statistical Planning and Inference*, 86(2):595–617, May 2000.

[106] D. R. Stinson and R. Wei. Combinatorial properties and constructions of traceability schemes and frameproof codes. *SIAM J. Discret. Math.*, 11(1):41–53, 1998.

[107] Gábor Tardos. Optimal probabilistic fingerprint codes. *J. ACM*, 55(2):1–24, 2008.

[108] TEKTRONIX. Mpeg-2 elementary streams corpus. URL: ftp://ftp.tek.com/tv/test/streams/Element/index.html.

[109] R.G. van Schyndel, A.Z. Tirkel, and C.F. Osborne. A digital watermark. *Image Processing, 1994. Proceedings. ICIP-94., IEEE International Conference*, 2:86–90 vol.2, Nov 1994.

[110] R. S. Veerubhotla, A. Saxena, V. P. Gulati, and A. K. Pujari. Gossip codes for fingerprinting: Construction, erasure analysis and pirate tracing. *Journal of Universal Computer Science*, 11(1):122–149, 2005.

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

[112] Giovanni Vigna. Cryptographic traces for mobile agents. In *Mobile Agents and Security*, pages 137–153, London, UK, 1998. Springer-Verlag.

[113] Andrew J. Viterbi. *CDMA: principles of spread spectrum communication*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1995.

[114] Andrew J. Viterbi and James K. Omura. *Principles of Digital Communication and Coding*. McGraw-Hill, Inc., New York, NY, USA, 1979.

[115] Branka Vucetic and Jinhong Yuan. *Turbo codes: principles and applications*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[116] Neal R. Wagner. Fingerprinting. In *SP '83: Proceedings of the 1983 IEEE Symposium on Security and Privacy*, page 18, Washington, DC, USA, 1983. IEEE Computer Society.

[117] Z. J. Wang, Min Wu, Hong Zhao, K. J. R. Liu, and W. Trappe. Resistance of orthogonal gaussian fingerprints to collusion attacks. In *ICME '03: Proceedings of the 2003 International Conference on Multimedia and Expo*, pages 617–620, Washington, DC, USA, 2003. IEEE Computer Society.

[118] Z.J. Wang, Min Wu, H.V. Zhao, W. Trappe, and K.J.R. Liu. Anti-collusion forensics of multimedia fingerprinting using orthogonal modulation. *Image Processing, IEEE Transactions on*, 14(6):804–821, June 2005.

[119] Bennet S. Yee. A sanctuary for mobile agents. In *Secure Internet Programming*, Lecture Notes in Computer Science, pages 261–273, London, UK, 1999. Springer-Verlag.

[120] Katsunari Yoshioka, Junji Shikata, and Tsutomu Matsumoto. Systematic treatment of collusion secure codes: Security definitions and their relations. In *Information Security, 6th International Conference, ISC 2003, Bristol, UK, October 1-3, 2003, Proceedings*, pages 408–421, 2003.

[121] Zhiguang Zhang, Xiaosu Chen, and Miao Zhou. A digital fingerprint coding based on turbo codes. *Computational Intelligence and Security, 2007 International Conference on*, pages 897–901, 2007.

[122] Yan Zhu, Wei Zou, and Xinshan Zhu. Collusion secure convolutional fingerprinting information codes. In *ASIACCS '06: Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, pages 266–274, New York, NY, USA, 2006. ACM Press.