Securing Digital Images

A thesis submitted in fulfillment of the requirements for the award of the degree

Doctor of Philosophy

from

UNIVERSITY OF WOLLONGONG

by

Chandrapal Kailasanathan

Computer Science Department
August 2003
© Copyright 2003

by

Chandrapal Kailasanathan

All Rights Reserved
Dedicated to
Prashanna and Karthiha
Declaration

This is to certify that the work reported in this thesis was done by the author, unless specified otherwise, and that no part of it has been submitted in a thesis to any other university or similar institution.

Chandrapal Kailasanathan
August 14, 2003
List of Publications


90%-Candidate 10%-Supervisors

- C.Kailasanathan, R.Safavi Naini, and P.ogunbona, ”Image Authentication Surviving Acceptable Modifications”, Proceedings of the (NSIP=01 2001) IEEE-EURASIP Workshop on Nonlinear Signal and Image Processing, University of Delaware, Baltimore, MD USA, June 3-6, 2001, CD-ROM.

95%-Candidate 5%-Supervisors

- C.Kailasanathan, R.Safavi Naini, and P.ogunbona ”Compression Performance of JPEG Encryption Scheme”, Proceedings of the IEEE-14th International Conference on Digital Signal Processing, University of Patras, Santorini, Greece, July 1-3,2002

100%-Candidate 0%-Supervisors


100%-Candidate 0%-Supervisors

and International Workshop on Multimedia Network Systems and Applications (MNSA 2003 Workshops) 19-22 May 2003 in Providence, Rhode Island, USA

85%-Candidate 15%-Supervisors


100%-Candidate 0%-Supervisors


85%-Candidate 15%-Supervisors
Abstract

With the advent of the Internet and World Wide Web, images are being transferred over insecure channels from different parts of the world. Those who receive images may not exactly know where they have originated from or who their sender was. In those circumstances, determining the authenticity of received images would be extremely difficult and therefore providing authenticity to images has been considered as an essential area of image security research. On the other hand, processing images for storage reduction, noise removal, or quality enhancement, is also essential in many computer applications, because images are inherently larger than text and are subject to noise or transcoding.

The authentication of image is achieved either by appending a Message Authentication Code or a Signature to an image or by embedding a watermark into the image which is fragile to any modification. On the other hand, storage reduction for images is mainly achieved by lossy compression techniques such as JPEG or MPEG, and the quality enhancement or noise reduction for images is accomplished by employing filtering techniques such as Median or Gaussian.

In this thesis, we propose hash functions which extract essential image features that survive acceptable level of JPEG compression while distinguishing other modifications using statistical measures of central tendency, dispersion, hypothesis of testing, and k-mean clustering, and wavelet/discrete cosine transform. For the proposed transform based hash functions, we optimize performance by fixing some essential parameters such as the size of hash, the value of the threshold and the low-pass filter used. We also propose two oblivious fragile watermarking schemes. The first one embeds a logo on feature points or critical points extracted using wavelet or discrete cosine transform, and the next one embeds a logo on image blocks based on polarity of pixel points. For the proposed schemes, after analysing the security level, the possibilities of embedding multiple logos for secret sharing have also been investigated.
The appendix of the thesis addresses some of our early work on combined compression and encryption scheme based on adaptive Huffman coding, compression performance of a proposed JPEG encryption scheme, and cryptanalysis of Huffman coding with four symbol alphabet.
I would like to thank Prof. Rei Safavi Naini, my supervisor, for her many suggestions and constant support during this research. I am also thankful to Dr. Philip Ogumbona for his guidance and help in the areas of image and signal processing through out my research. Especially, the k-mean software given by him was very useful in producing attractive results in feature extraction of images. The suggestions given by Dr. Wanqing Li is also very much appriciated.

The work, which had been done by Dr. Takeyuki Uehara in the areas of arithmatic coding and wavelet transform, in order to provide security to JPEG 2000 was inspiring, and led us to new ideas and thouhts. I am also thankful to Dr. Gareth Brisbane for helping me to find the object code of Stirmark, which enabled me to manipulate images without designing software for each image processing modification. I would also like to thank Prof. Ronald L.Rivest and Dr. Sonja Grgic for granting me permission to use some of their work in this thesis.

I should also mention that my graduate studies in Wollongong were supported by the Australian Research Council and Motorola Australian Research Centre.

Of course, I am grateful to my parents for giving me a good education and my family members for their patience and love. Without them this work would never have come into existence (literally).

Finally, I wish to thank the fellow students and staff of the Department of Computer Science, University of Wollongong for providing me all the facilities and a stimulating environment to carry out this research.

Wollongong, Australia
Chandrapal Kailasanathan
July 2001
Contents

List of Publications v
Abstract vii
Acknowledgements ix

1 Introduction
1.1 Motivation ........................................ 1
1.2 Objective ......................................... 3
1.3 Contributions ...................................... 4
1.3.1 My Contribution ................................ 5

2 Information Security and Image Compression 7
2.1 Introduction ....................................... 7
2.2 Encryption and Decryption ....................... 9
2.2.1 Basic Terminology and Concepts .............. 10
2.2.2 Symmetric-key Encryption .................... 13
2.2.3 Public-key Cryptography ..................... 14
2.2.4 Symmetric-key vs. Public-key Cryptography . 14
2.2.5 Attacks on Encryption Schemes ............... 15
2.3 Authentication: Cryptographic vs. Image ....... 16
2.3.1 Cryptographic Authentication ................ 16
2.3.2 Watermarking for Image Authentication .... 17
2.3.3 Attacks on Cryptographic Authentication Schemes .... 18
2.3.4 Attacks on Fragile Watermarking Schemes ... 18
2.4 JPEG Compression ................................. 19
2.4.1 JPEG: DCT-Based Image Coding ............. 19
2.4.2 JPEG-2000: Image Compression Standard ... 23
2.4.3 Compression vs. Filtering .................... 23
2.5 Combined Compression and Crypto Systems .... 24

x
2.5.1 Compression Surviving Authentication ........................................ 24
2.5.2 Combined Compression and Encryption .................................... 25
2.6 Conclusion .................................................................................. 25

3 Cryptographic Hash Functions and Data Integrity .......................... 26
3.1 Introduction ................................................................................ 26
3.2 Usage of Hash Functions .............................................................. 27
3.3 General Classification ................................................................. 27
3.4 Basic Properties and Definitions .................................................. 28
3.4.1 Attacks on Hash functions and MACs ...................................... 30
3.5 Relationship between Properties .................................................. 31
3.6 Other Hash Function Properties and Applications ....................... 32
3.7 Data Integrity ............................................................................. 32
3.7.1 Data Integrity ......................................................................... 32
3.7.2 Data Origin Authentication (Message Authentication) ............ 33
3.7.3 Transaction Authentication ..................................................... 33
3.7.4 Models for Providing Data Integrity ....................................... 34
3.8 Image Hashing for Authentication ............................................... 36
3.9 Image Hashing .......................................................................... 37
3.10 Previous Work ......................................................................... 38
3.10.1 A Robust Image Authentication Method Distinguishing JPEG 
Compression from Malicious Manipulation-By Ching-Yung Lin 
and Shih-Fu Chang .......................................................................... 39
3.10.2 Towards Robust Content Based Technique for Image Authenti-
cation by Maria Paula Queluz ......................................................... 41
3.10.3 Compression Tolerant Image Authentication by Sushil Bhattachar-
jee and Martin Kutter ..................................................................... 43
3.10.4 Applying Signatures on Digital Images by I.Pitas and T.H.Kaskalis 45
3.10.5 A Robust Content Based Digital Signature for Image Authenti-
cation by Marc Schneider and Shih-Fu Chang ................................ 48
3.10.6 Content-based Digital Signature for Motion Picture Authentica-
tion and Content-Fragile Watermarking by Jana Dittmann, Arnd 
Steinmetz, and Ralf Steinmetz ....................................................... 50
3.11 Conclusion .............................................................................. 52

4 Statistical Based Image Hashing .................................................. 53
4.1 Introduction .............................................................................. 53
4.1.1 Statistics .................................................. 53
4.1.2 Hypothesis Testing ........................................ 54
4.1.3 K-mean/LBG Algorithm ................................. 54
4.2 Image Hash .................................................. 55
4.2.1 Hash Generation and Verification ....................... 55
4.2.2 Method-1: Block Statistics .............................. 55
4.2.3 Possible Attacks ......................................... 56
4.2.4 Method-2: Region Statistics ............................. 58
4.2.5 A Comparison ............................................. 62
4.2.6 Method-3: K-means ....................................... 62
4.2.7 Method-4: Hypothesis of Testing ....................... 64
4.2.8 Method-5: Edge Detection .............................. 66
4.2.9 Performance Analysis .................................... 67
4.3 Conclusion ................................................... 70

5 Critical Set Based Image Hashing 72
5.1 Introduction ................................................ 72
5.1.1 Wavelet Transform ..................................... 73
5.1.2 Discrete Cosine Transform .............................. 76
5.1.3 Edge Detection .......................................... 77
5.2 Image Hash ................................................ 78
5.2.1 Basic Description of the Scheme ....................... 78
5.2.2 Hash Generation Algorithm ............................. 78
5.2.3 Verification Algorithm .................................. 79
5.2.4 Length of Hash: Measure of Security ................. 79
5.3 Experimental Results ..................................... 80
5.3.1 Constant Value for $\rho$ within One Scale .......... 80
5.3.2 Constant Value for $\rho$ within Scale One and Two ... 84
5.3.3 A Comparison with Edge Detection .................... 88
5.3.4 Fixed Size Hash ........................................ 91
5.3.5 Invariance of the Critical Set to Different Methods . 95
5.3.6 Optimizing AC Coefficient Addition .................. 98
5.3.7 Performance in High-High Pass Band ................. 102
5.3.8 Real World Attacks .................................... 103
5.4 Security Analysis .......................................... 107
5.4.1 $\rho$ as a Key ........................................... 107
5.4.2 Compression Levels as Keys .................................. 107
5.4.3 Low-pass Filter as a Key .................................. 107
5.5 Size of Key Space ............................................. 108
5.5.1 \( \rho \) as a Key ........................................... 108
5.5.2 Compression Levels as Keys ................................. 108
5.5.3 Low-pass Filter as a Key .................................. 108
5.6 A Comparison ............................................... 108
5.6.1 Wavelet ................................................... 109
5.6.2 Scales and Size of Filters ................................. 109
5.6.3 Computations Involved ................................... 109
5.6.4 Threshold Selection/Compression Tolerance ............. 110
5.6.5 Redundant Information/Orientation ....................... 110
5.7 Conclusion .................................................. 111

6 Fragile Watermark on Critical Points .............................. 112
6.1 Introduction .................................................. 112
6.2 Previous Schemes ........................................... 112
6.2.1 Checksum Technique ..................................... 112
6.2.2 M-Sequence Approach ................................... 113
6.2.3 Two Dimensional Spatial Watermarks ..................... 113
6.2.4 DCT Coefficient Modulation ............................. 114
6.2.5 Sub-Band Watermarking Technique ...................... 115
6.2.6 A Visible Watermark-Young-Mintzer Scheme ........... 116
6.2.7 Feature Based Watermark-Bhattacherjee’s Scheme ...... 117
6.3 A New Fragile Watermarking Scheme ......................... 118
6.3.1 Our Proposal ............................................. 118
6.3.2 Watermark Embedding ................................... 119
6.3.3 Verification Algorithm ................................... 120
6.4 Experimental Results ....................................... 120
6.4.1 Accuracy of Critical Points Extraction .................... 120
6.4.2 Sensitivity to Compression Levels ....................... 122
6.4.3 Imperceptibility vs. Amount of Information Hidden .... 122
6.5 Threshold Selection and Reconstruction of Images .......... 123
6.6 Synchronization Loss ........................................ 124
6.7 Advantages .................................................. 124
6.8 Drawback and Possible Improvements ......................... 125
6.9 Possible Extensions: Multiple Watermarks ............................................ 126
  6.9.1 Threshold Based ................................................................. 126
  6.9.2 Region Based ................................................................. 126
  6.9.3 A Comparison between Threshold and Region Based Methods .............. 127
6.10 Security Analysis ............................................................................ 127
  6.10.1 $\rho$ as a Key ................................................................. 128
  6.10.2 Compression Levels as Keys ................................................... 128
  6.10.3 Low-pass Filter as a Key ....................................................... 129
  6.10.4 Binary Look Up Table as a Key .............................................. 129
  6.10.5 Centroids as Keys .................................................................. 131
6.11 Size of Key Space ........................................................................... 131
  6.11.1 $\rho$ as a Key ................................................................. 131
  6.11.2 Compression Levels as Keys ................................................... 131
  6.11.3 Low-pass Filter as a Key ....................................................... 131
  6.11.4 Binary Look Up Table as a Key .............................................. 132
6.12 A Comparison Between Bhattacharjee’s Scheme and Ours ...................... 132
6.13 Performance Evaluation .................................................................... 133
  6.13.1 Edit Distance as a Measure of Distortion ..................................... 134
  6.13.2 Statistical Hypothesis Testing ................................................... 135
  6.13.3 Real World Attack ............................................................... 136
  6.13.4 Type 1 and Type 11 Errors ...................................................... 137
6.14 Conclusion ....................................................................................... 140

7 Fragile Watermark Based on Polarity of Pixel Points ............................... 145
  7.1 Introduction ............................................................................... 145
  7.2 Previous Schemes ........................................................................ 145
  7.3 Our Proposal ............................................................................... 146
    7.3.1 Watermark Embedding ....................................................... 147
    7.3.2 Verification Algorithm ....................................................... 147
  7.4 Experimental Results .................................................................... 148
  7.5 Drawbacks and Possible Remedies ................................................ 149
  7.6 Advantages of the Scheme .......................................................... 150
  7.7 Possible Improvements: Multiple Watermarks .................................... 150
    7.7.1 Variance Based ................................................................... 150
    7.7.2 K-mean Based .................................................................... 150
    7.7.3 A Comparison between Variance and K-means Based Methods ....... 151

xv
7.8 Security Analysis ......................................................... 152
  7.8.1 Block Size as a Key ............................................. 152
  7.8.2 Seed of PRNG as a key ......................................... 152
  7.8.3 Binary Look Up Table as a Key ............................... 153
  7.8.4 Centroids as Keys ................................................ 153
7.9 Size of Key Space ..................................................... 154
  7.9.1 Block Sizes as Keys ............................................. 154
  7.9.2 Seed of a PRNG as a Key ..................................... 154
  7.9.3 Binary Look Up Table as a Key ............................... 155
7.10 A Comparison Between Bhattacharjee’s Scheme and Ours ............. 155
7.11 Performance Evaluation ............................................. 156
7.12 Conclusion ............................................................ 156

8 Conclusion ............................................................... 158
  8.1 Proposed Schemes .................................................. 158
  8.2 Open Problems ....................................................... 158

A Compression and Encryption ........................................... 160
  A.1 Securing Adaptive Huffman Coding .............................. 160
    A.1.1 Introduction .................................................. 160
    A.1.2 Dynamic Huffman Coding ................................... 161
    A.1.3 Huffman Coding Encryption Schemes ....................... 162
    A.1.4 Securing Dynamic Huffman Encryption System .......... 165
    A.1.5 Experiment .................................................... 167
    A.1.6 Conclusion .................................................... 167
  A.2 Critical Analysis of MPEG Encryption Schemes ................. 168
    A.2.1 Introduction .................................................. 168
    A.2.2 Incorporation of Compression and Encryption ............ 168
    A.2.3 MPEG Encryption Algorithm ................................ 169
    A.2.4 Zig-Zag Permutation Algorithm ............................ 170
    A.2.5 Video Encryption Algorithm ................................ 172
    A.2.6 Sign-bit Encryption .......................................... 173
    A.2.7 Conclusion .................................................... 174
  A.3 Compression Performance of JPEG Encryption Scheme ............ 175
    A.3.1 Introduction .................................................. 175
    A.3.2 Dynamic Huffman Coding .................................... 175
    A.3.3 JPEG Encryption Scheme ..................................... 176

xv
A.3.4 Zig-Zag Permutation Algorithm .......................... 177
A.3.5 Compression Performance ................................. 178
A.3.6 Conclusion .................................................... 179
A.4 On Breaking Huffman Codes with Four Symbols ............ 182
A.4.1 An Overview of the Problem ............................... 182
A.4.2 Four Symbol Source Alphabet ............................ 183
A.4.3 Conclusion .................................................... 190

Bibliography ......................................................... 191
List of Tables

3.1 Sobal masks ............................................................. 42
3.2 Two images having a digonal edge with different intensities ........ 42
3.3 Gradients computed ................................................... 42
3.4 Two distinct images .................................................. 49
3.5 Histogram computed .................................................. 50
4.1 Size of hash .............................................................. 64
4.2 $T_n$ using maximum of mean and standard deviation on 1200, 300, and
12 square blocks ......................................................... 70
4.3 $D_m$ using mean and standard deviation on 15 and 30 regions ........ 71
4.4 ED on 5 different starting code books .................................. 71
5.1 The horizontal and vertical convolution masks for edge detection .... 77
5.2 The number of co-ordinates in a critical set and the matching percentages
on modified images for wavelets DB-3, DB-6, DB-10, and BO-(4,4), and
XV’s edge detector ..................................................... 81
5.3 The number of co-ordinates in a critical set and the matching percentages
on modified images for wavelets DB-3, DB-6, DB-10, and BO-(4,4), and
XV’s edge detector ..................................................... 81
5.4 The number of co-ordinates in a critical set and the matching percentages
on modified images for wavelets DB-3, DB-6, DB-10, and BO-(4,4), and
XV’s edge detector ..................................................... 82
5.5 The number of co-ordinates in a critical set and the matching percentages
on modified images for wavelets DB-3, DB-6, DB-10, and BO-(4,4), and
XV’s edge detector ..................................................... 82
5.6 The number of co-ordinates in a critical set and the matching percentages
on modified images for wavelets DB-3, DB-6, DB-10, and BO-(4,4), and
XV’s edge detector ..................................................... 83
5.7 The number of co-ordinates in a critical set and the matching percentages on modified images for wavelets DB-3, DB-6, DB-10, and BO-(4,4), and XV’s edge detector .................................................. 85
5.8 The number of co-ordinates in a critical set and the matching percentages on modified images for wavelets DB-3, DB-6, DB-10, and BO-(4,4), and XV’s edge detector .................................................. 85
5.9 The number of co-ordinates in a critical set and the matching percentages on modified images for wavelets DB-3, DB-6, DB-10, and BO-(4,4), and XV’s edge detector .................................................. 86
5.10 The number of co-ordinates in a critical set and the matching percentages on modified images for wavelets DB-3, DB-6, DB-10, and BO-(4,4), and XV’s edge detector .................................................. 86
5.11 The number of co-ordinates in a critical set and the matching percentages on modified images for wavelets DB-3, DB-6, DB-10, and BO-(4,4), and XV’s edge detector .................................................. 87
5.12 The number of co-ordinates in a critical set and the matching percentages on modified images for Canny ............................................. 89
5.13 The number of co-ordinates in a critical set and the matching percentages on modified images for Canny ............................................. 89
5.14 The number of co-ordinates in a critical set and the matching percentages on modified images for Canny ............................................. 89
5.15 The number of co-ordinates in a critical set and the matching percentages on modified images for Canny ............................................. 90
5.16 The number of co-ordinates in a critical set and the matching percentages on modified images for Canny ............................................. 90
5.17 The number of co-ordinates in a critical set and the matching percentages on modified images for wavelets DB-3, DB-6, DB-10, and BO-(4,4), Canny’s edge detectors, and DCT ........................................... 92
5.18 The number of co-ordinates in a critical set and the matching percentages on modified images for wavelets DB-3, DB-6, DB-10, and BO-(4,4), Canny’s edge detector, and DCT ........................................... 92
5.19 The number of co-ordinates in a critical set and the matching percentages on modified images for wavelets DB-3, DB-6, DB-10, and BO-(4,4), Canny’s edge detector and DCT ........................................... 93
5.20 The number of co-ordinates in a critical set and the matching percentages on modified images for wavelets DB-3, DB-6, DB-10, and BO-(4,4), Canny’s edge detector, and DCT .......................... 93

5.21 The number of co-ordinates in a critical set and the matching percentages on modified images for wavelets DB-3, DB-6, DB-10, and BO-(4,4), Canny’s edge detector and DCT .......................... 94

5.22 The number of critical points common to different methods ........ 95
5.23 The number of critical points common to different methods ........ 95
5.24 The number of critical points common to different methods ........ 96
5.25 The number of critical points common to different methods ........ 96
5.26 The number of critical points common to different methods ........ 96
5.27 Matching percentage for modified images of Lena ......................... 100
5.28 Matching percentage for modified images of Peppers ...................... 100
5.29 Matching percentage for modified images of Pills ......................... 100
5.30 Matching percentage for modified images of Paper ....................... 101
5.31 Matching percentage for modified images of Corrosion ................... 101
5.32 Matching percentage for modified images of Lena ....................... 102
5.33 Matching percentage for modified images of Peppers ..................... 103
5.34 Matching percentage for modified images of Paper ....................... 103
5.35 Matching percentage for modified images of Pills ......................... 104
5.36 Matching percentage for modified images of Corrosion ................... 104
5.37 Acceptance range \([a,b]\) and \(\beta\) ............................................. 106
5.38 The number of critical points common to different combination of compression levels ................................................................. 107

6.1 Size of logo embedded vs. imperceptibility ................................. 123
6.2 The number of critical points common to different combination of compression levels ................................................................. 128
6.3 The number of critical points common to different methods of lowpass filterings ................................................................. 129
6.4 Look up tables (LUT), PSNR values of watermarked image and extracted logo, size of critical set before \(|CSB|\) and after embedding \(|CSA|\) and the number of common points, and the \(\rho\) used .................. 130
6.5 Edit Distances between embedded and extracted logos, size of logo embedded, and size of logo extracted ................................. 134
6.6 Edit Distances between embedded and extracted logos .................. 136
6.7 PSNR, MSE, embedded and extracted string sizes, and the Edit Distances are shown .................................................... 137
6.8 Type 1 and Type 11 Errors .................................................... 138
6.9 Edit Distances and PSNR between images ............................... 139
6.10 Critical points common to both original and watermarked images considering compression quality levels of 70, 50, and 25 .................. 141
6.11 Critical points common to both original and watermarked images without considering compression ........................................ 142
7.1 PSNR computed on images choosing five different block sizes .... 148
7.2 Look up tables, PSNR values of watermarked image and extracted logo 153
A.1 Compression ratios for JPEG without permutation ................. 180
A.2 Compression ratios after each block’s zig-zag ordered DCT coefficients have been permuted with one of four different permutations .......... 180
A.3 Compression ratios for JPEG with adaptive Huffman coding ......... 181
A.4 Transition probabilities in the CPM or Transition matrix ............... 187
## List of Figures

2.1 Two-party communication using encryption, with a secure channel for key exchange. The decryption key \( d \) can be efficiently computed from the encryption key \( e \) .......................... 13

2.2 Encryption using public-key techniques ............................... 14

2.3 DCT-based encoder and decoder processing steps ..................... 20

2.4 Preparation of quantized coefficients for entropy coding ............. 22

3.1 Three methods for providing data integrity using hash functions ...... 36

4.1 Comparison of thresholds ................................................. 66

4.2 \( \alpha, \beta \), and the acceptance region \([a, b] \) assuming the values follow normal distributions with means \( \bar{c} \) and \( \bar{o} \) and standard deviations \( \sigma_c \) and \( \sigma_o \) respectively. .................................................. 69

5.1 Critical co-ordinates for images (a) Lena (DB-3, DB-6, DB-10, BO-(4,4), DCT) (b) Peppers (DB-3, DB-6, DB-10, BO-(4,4), DCT) (c) Pills (DB-3, DB-6, DB-10, BO-(4,4), DCT) (d) Paper (DB-3, DB-6, DB-10, BO-(4,4), DCT) and (e) Corrosion (DB-3, DB-6, DB-10, BO-(4,4), DCT) 97

5.2 \( \alpha, \beta \), and the acceptance region \([a, b] \) assuming the values follow normal distributions with means \( \bar{c} \) and \( \bar{o} \) and standard deviations \( \sigma_c \) and \( \sigma_o \) respectively. ........................... 105

6.1 (a) Original Logo (b) Logo on Lena (c) Logo on Peppers (d) Logo on Pills (e) Logo on Paper (f) Logo on Corrosion .......................... 121

6.2 (a) and (b) are embedded logos and (c) and (d) are extracted logos . 126

6.3 (a) Three Voronoi regions chosen (b) three embedded logos and (c) three extracted logos ........................................... 127
6.4 (a) Original logo and the logo extracted from the watermarked image when (b) random LUT (c) LUT with 0101.. (d) LUT with 1010.. (e) LUT with 0011.. (f) LUT with 1100.. (g) LUT with 000111 and (h) LUT with 111000 are used ................................. 130
6.5 Logo recovered from (a) JPEG compression at quality level 90 (b) horizontal flipping (c) oilpainting and (d) vertical flipping .................. 134
6.6 Original car image, watermarked car image and the modified watermarked car image .................................................. 143
6.7 Manipulations without disturbing the critical points ................. 144
6.8 Modified images .......................................................... 144

7.1 (a) Original Logo (b) Logo on Lena (c) Logo on Peppers (d) Logo on Pills (e) Logo on Paper (f) Logo on Corrosion ...................... 149
7.2 (a) and (b) are embedded logos and (c) and (d) are extracted logos .. 151
7.3 (a) Three Voronoi regions chosen (b) embedded logos and (c) extracted logos ...................................................... 152
7.4 (a) Original logo and the logo extracted from watermarked image when (b) random LUT (c) LUT with 0101.. (d) LUT with 1010.. (e) LUT with 0011.. (f) LUT with 1100.. (g) LUT with 000111 and (h) LUT with 111000 are used ................................. 154
7.5 (a) cutting-1 (b) cutting-2 (c) copy and paste-1 and (d) copy and paste-2 attacks on Peppers image ................................. 156
7.6 Logo recovered from (a) JPEG compression at quality level 90 (b) blurring (c) embossing (d) horizontal flipping (e) oilpainting (f) rotation (g) vertical flipping (h) sharpening (i) cutting-1 (j) cutting-2 (k) copy and paste-1 and (l) copy and paste-2 ................................. 157

A.1 Code tree .............................................................. 164
A.2 (a) Original Lenna (b) Lenna after encryption and (c) Lenna after decryption with a wrong key ........................................ 167
A.3 All possible Huffman trees with four symbols ...................... 184
A.4 A Huffman tree and its Markov chain .............................. 186
A.5 A Huffman tree and its Markov chain ................................ 187
Chapter 1

Introduction

With the invention of the Internet and World Wide Web, images are being widely used for various purposes. Video on demand and video conferencing systems store video images for users who have paid for the services. A vast number of fingerprint images are being stored for investigation purposes. Images are being transmitted from sensing devices for medical and astronomical needs. Libraries store large collection of important images for historical records. Printing industry processes vast amount of images for publishing purposes. All these applications allow extensive access to and reuse of visual materials. This has lead to unauthorized copying, reading, manipulating or removing of image data. Therefore, preserving the authenticity and integrity of images has become an important area of research in multimedia information security.

1.1 Motivation

With the rapid expansion of the Internet, controlling access to images and protecting the authenticity and integrity of images have become an essential part of computer security research. Proliferation of image manipulation softwares such as Photo Shop for image enhancement and noise removal has made tampering with images without being visually noticed an easy task. The two common methods that promote protection of images from tampering are encryption and authentication. While encryption scrambles the original image into a cipher image for providing safety, authentication provides the reliability of the source from which the image has been originated.

Authenticity to images can be provided either by annexing a tag to the image or by embedding a secret code or a watermark into the image itself. The tag attached for authentication purposes is generally known as Message Authentication Code (MAC) or Digital Signature (DS). MACs are hash values generated by hash functions with an additional input, which is a secret key and will be known only to the sender and the receiver of the image. Hash values by themselves can not guarantee security. To provide
security guarantee, the hash value can be signed by a public key encryption scheme to produce a digital signature or can be encrypted by a symmetric key encryption function.

Because of the importance of image compression for storage efficiency, providing authenticity to compressed images has become an important problem. Because compression changes pixel values while preserving the 'visual look' of the image, using traditional cryptographic authentication schemes to provide authenticity to images is not possible. This is because a hash (MAC) of the original image will not be the same as the hash (MAC) of the compressed version of the image and so the authentication system will fail to authenticate such compressed images. A number of compression tolerant authentication schemes have been proposed [7, 38, 39, 40, 36, 45, 37]. Some of these schemes have already been broken, while others are inefficient in terms of security and complexity. One of the well known schemes [38, 39, 40] has already been broken [41]. This indicates that a clear security and efficiency assessment of all proposed schemes are necessary. It also shows providing authenticity to compressed images is a difficult and tedious task.

Although above mentioned authentication schemes provide sufficient authenticity to images, since the tags are appended to the image, it is always possible for the tags to be removed while they are in transmission. To avoid this sort of attacks, more efficient authentication schemes have been proposed. These schemes embed a key dependent secret codes or watermarks into the image. This is by modifying the pixel values without considerably degrading the image 'visual look'. Watermarking schemes used for image authentication must be fragile, which means the watermark must disappear if the image is tempered with.

A number of fragile watermarking schemes [65, 66, 67] have been proposed. Some are broken while the others are without clear security assessment. None of these schemes have been extended to multiple watermark embedding. This indicates that a clear analysis of the existing schemes is necessary to provide sufficient protection against security vulnerability, and to extend the existing schemes to come up with multiple watermarking schemes for images.

Among all image processing needs, compression of image data for efficient storage and transmission plays a vital role, since image files are inherently larger than text files. A typical color image could contain 1 M-byte of data. For videos with the rate of 30 frames per second, the situation is even worse. These examples indicate how large the image files are. To provide efficient storage for images, several standards are proposed.
with JPEG (Joint Photographic Experts Group) and MPEG (Moving Picture Expert Group) being the most widely used ones for still and moving images respectively. The detail description of JPEG and MPEG standards can be found in [26] and [5].

These standards provide the specification for conversion between source image data and compressed image data, and the guidelines for implementation. The whole compression process can be categorised into two different classes, lossy and lossless. The lossless compression part uses DCT (Discrete Cosine Transform) or WT (Wavelet Transform) for frequency decomposition, and Huffman or Arithmetic coding for entropy coding. The lossy compression part uses quantization for data reduction.

Processing images for quality enhancements is another important area in image processing applications. The quality enhancements for images are mainly achieved by filtering techniques. These processing can roughly be classified into sharpening or smoothing of images. For our security and authenticity needs, these processing are considered as attacks. There are also other types of attacks, such as cropping, rotation, removal or substitution. These are known as geometric attacks of an image.

1.2 Objective

The goal of this thesis is to investigate the existing image authentication schemes and then to propose schemes that provide protection against tampering with the image. This is mainly achieved by proposing hash functions which extract essential image features that survive acceptable level of JPEG compression while distinguishing other modifications (smoothing, sharpening, cropping, rotation, removal or substitution) using statistical measures of central tendency, dispersion, hypothesis of testing, k-mean clustering, and wavelet or discrete cosine transform and then by optimizing detection performance by fixing some essential parameters such as the size of hash, the value of the threshold used, or the low-pass filter used.

Another goal is to study some of the well known fragile watermarking schemes and then to propose new and sophisticated fragile watermarking schemes. This is mainly achieved by embedding logos or watermarks either on feature points extracted using wavelet or discrete cosine transform or on image blocks based on polarity of pixel points, by analysing the security level of the overall method, and by investigating the possibility of embedding multiple logos or watermarks for secret sharing.
1.3 Contributions

In this thesis, we propose hash functions which extract essential image features that survive acceptable level of JPEG compression while distinguishing other modifications, firstly by using statistical measures of central tendency, dispersion, hypothesis of testing, and k-mean clustering, and secondly by using transformations such as wavelet or discrete cosine transform. For the proposed transform based hash functions, we optimize performance by fixing some essential parameters such as the size of hash, value of the threshold or low-pass filter used.

We also propose two fragile watermarking schemes. The first one embeds a logo on feature points extracted using wavelet or discrete cosine transform and the next one embeds a logo on image blocks based on polarity of pixel points. For the same schemes, after analysing the security level of the overall method, the possibilities of embedding multiple logos for secret sharing have also been investigated.

The first part of the appendix addresses some of our early work on combined encryption and compression schemes by proposing a new combined encryption and compression scheme which incorporates some encryption techniques within adaptive Huffman encoding algorithm. It also provides protection measures against the security vulnerability, after exploring the proposed attacks on adaptive arithmetic coding encryption scheme. The second part of the appendix, reviews some proposed encryption schemes for JPEG/MPEG, and then investigates the compression performance of JPEG encryption based on permutation. The last part of the appendix extends the crypt-analysis proposed for Huffman sequence with three symbol source alphabet to four symbol source alphabet.

This thesis is organized as follows. Chapter 1 gives the introduction to thesis. Chapter 2 explains what information security and image compression are. Chapter 3 defines cryptographic hash functions and explores how data integrity is achieved via hash functions. In Chapter 4, we propose hash functions, using basic statistical measures, hypothesis testing and K-mean segmentation. In Chapter 5, we propose hash functions based on Wavelet and Discrete Cosine Transforms. Chapters 6 proposes a fragile watermarking scheme which embeds logos on feature points extracted using Wavelet/Discrete Cosine Transform. Chapter 7 proposes another fragile watermarking scheme which embeds logos on image blocks based on polarity of image pixel. Finally, Chapter 8 concludes the thesis. Appendix Chapter 1, proposes a scheme for securing adaptive Huffman coding. Appendix Chapter 2, critically analyses proposed MPEG encryption schemes. Appendix Chapter 3, investigates the compression performance of
JPEG encryption scheme. Appendix Chapter 4, extends the crypt-analysis of Huffman coding with 3 symbol alphabets to 4 symbol alphabets.

1.3.1 My Contribution

Since this piece of work has been carried out as a member of a team of doctoral and post doctoral candidates under the major guidance of Prof. Rei Safavi-Naini and Dr. Philip Ogumbona, it would be wise to let the reader know the proportion of the work that has been contributed by the author.

- Chapter 1 Introduction
  Background chapter

- Chapter 2 Information Security and Image Compression
  Background chapter

- Chapter 3 Cryptographic Hash Functions and Data Integrity
  Background chapter, 30%-Author, and 5%-Supervisors

- Chapter 4 Statistical Based Image Hashing
  95%-Author, and 5%-Supervisors

- Chapter 5 Critical Set Based Image Hashing
  85%-Author, and 15%-Supervisors

- Chapter 6 Fragile Watermark on Critical Points
  100%-Author

- Chapter 7 Fragile Watermark Based on Polarity of Pixel Points
  100%-Author

- Chapter 8 Conclusion
  Concluding chapter

- Appendix Chapter 1 Compression and Encryption
  90%-Author, and 10%-Supervisors

- Appendix Chapter 2 Critical Analysis of MPEG Encryption Schemes
  Background chapter

- Appendix Chapter 3 Compression Performance of JPEG Encryption Scheme
  100%-Author
• Appendix Chapter 4 On Breaking Huffman Codes with Four Symbols
  100%-Author
Chapter 2

Information Security and Image Compression

2.1 Introduction

Information security or Cryptography is the study of providing confidentiality, data integrity, entity authentication, and data origin authentication by means of employing mathematical techniques such as number theory, group theory, statistical techniques etc.,. The goals of information security can broadly be classified into (1) privacy or confidentiality, (2) data integrity, (3) authentication and (4) non-repudiation.

1. Confidentiality is a service that provides access to the content of information only to the authorized. The terms secrecy, confidentiality and privacy have the synonymous meaning.

2. Data integrity refers to the quality of being unmanipulated. To determine data integrity, data manipulation by unauthorized parties must be detected. Data manipulation includes insertion, deletion or substitution of data.

3. Authentication and identification are terms closely related to each other. Two communicating parties should identify each other before entering into communication. Information transmitted over an in-secured channel should be authenticated with respect to origin, data of origin, data content, time sent, etc.

4. Non-repudiation is a service which prohibits an entity from refusing to accept earlier actions or commitments. A third party is needed to resolve the disputes arising from non-repudiation.

The fundamental goal of cryptography is to adequately address these four areas by providing prevention and detection of cheating and other malicious activities using a number of basic cryptographic tools (primitives) such as encryption schemes, hash functions, and digital signature schemes.
Using traditional cryptographic tools to provide protection against copying and cheating of images is not very efficient because image data is inherently large and redundant, and secret information such as signatures can be embedded on the image without degrading the perceptual quality. For these reasons, more efficient scheme such as \textit{watermarking} has been introduced by both cryptographic and image processing researcher. Though watermarking considerably differs from traditional cryptographic techniques, both methods have a common goal of protection against piracy. Different classes of watermarking schemes exist \cite{2}. They are known as \textit{watermarking for copyright protection}, \textit{fingerprinting for traitor tracking}, \textit{watermarking for copy protection}, and \textit{watermarking for image authentication}. For more detail description of these schemes consult \cite{2}.

Encryption schemes can generally be divided into two main divisions, which are known as \textit{symmetric key encryption schemes} and \textit{public key encryption schemes}. Authentication of images involves \textit{signing} or encryption of the digest of an image or embedding an imperceptible watermark signal on selected image pixels. The digest of an image is normally obtained by applying \textit{hashing} techniques. The level of secrecy is assessed on how well the crypto system is protected against attacks. The attacks on encryption schemes are widely known as \textit{ciphertext-only}, \textit{known plaintext}, \textit{chosen plaintext}, and \textit{chosen ciphertext} attacks. The attacks known for cryptographic authentication schemes are \textit{key recovery}, \textit{existential forgery}, \textit{selective forgery}. The attacks known for watermarking for authentication are \textit{blind}, \textit{image transfer}, \textit{mark transfer}, and \textit{key recovery}. We will describe all these terminologies in detail.

Image data uses huge storage space, large transmission band width and long transmission time. For these reasons, image data needs to be compressed. The compression of images is achieved by removing the redundancy present in images. These redundancies are identified as \textit{spatial}, \textit{spectral}, and \textit{temporal} redundancies. Spatial redundancy is due to correlation between neighboring pixel values, spectral redundancy is due to correlation between color planes or spectral bands, and temporal redundancy is due to correlation between adjacent frames in a sequence of images (videos).

Image compression for still images removes the spatial and spectral redundancies as much as possible by following the three main steps given below:

1. Transformation (Discrete Cosine or Wavelet Transform)

2. Quantization and

3. Entropy coding (Huffman or Arithmetic Coding)
Image transform decorrelates an image into frequency components or sub-bands. The transformed co-efficients are quantized and then entropy coded to produce a compressed sequence. These steps can roughly be divided into lossy and lossless compression techniques. In lossy compression some amount of information is lost, whereas in lossless compression the same information is preserved even after compression. Among the above three steps, since transformation and entropy coding do not involve in any lose of information, they can be classified as lossless, and since quantization involves in lose of information it can be classified as lossy.

JPEG and MPEG, which are the widely known standards for still and moving pictures, respectively, basically use the above three steps in their compression process. Although it wouldn’t be possible to describe the entire standard here, we will try to give the essential steps involved in DCT based image compression later in this chapter.

Since lossy compression techniques such as JPEG changes the original image data by removing some amount of redundant information, using traditional cryptographic methods to provide secrecy or authenticity to decompressed compressed images would be inefficient or impossible. Therefore, combining authentication or secrecy with compression has become essential in image processing and cryptology research.

Since compression surviving authentication is mainly achieved by hashing techniques, we will propose hash functions (or feature extraction schemes) or MACs (Message Authentication Codes) based on statistical (mean, standard deviation, and k-means) or image transformation (wavelet and discrete cosine transforms) methods. Because authentication of images is also achieved by watermarking techniques, we will also propose efficient fragile watermarking schemes (watermarking for authentication). An early work on combined compression and encryption scheme using adaptive Huffman coding has also been included in the appendix.

## 2.2 Encryption and Decryption

Encryption schemes are of two kinds, namely private-key encryption scheme and public-key encryption scheme. In private-key encryption scheme sender and receiver share the same key. Hence the private key encryption scheme is also known as symmetric key encryption scheme. Public-key encryption scheme has two distinct keys. One is known as public key, while the other is known as private key. Since both the keys are distinct, it is also known as asymmetric key encryption scheme. As there are two different keys, it is possible to make either the encryption or decryption key public. Suppose the encryption key is public, anyone can encrypt the message, but only the receiver can
decrypt cryptogram. This protects the secrecy of the message. On the other hand, if the decryption key is public, any one can decrypt the cryptogram, but only the owner of the encryption key can generate meaningful cryptograms. This protects the authenticity of the message.

2.2.1 Basic Terminology and Concepts

In this section we define the terminology needed for describing encryption schemes.

Encryption Domains and Codomains

The encryption domains and codomains are defined as below.

- Let a finite set $A$ be defined as the *alphabet*. For example, $A = \{0, 1\}$ be the *binary alphabet*. Note that any alphabet including a set of valid pixel values can be encoded in terms of the binary alphabet.

- Let a finite set $M$ be defined as the *message/image space*. $M$ consists of strings of symbols from an alphabet. *Plaintexts* are the elements of set $M$. For example, $M$ may consist of binary strings, images, English text etc.

- Let a finite set $C$ be defined as the *ciphertext space*. $C$ may be defined over an alphabet which is different from the alphabet for $M$. *Ciphertexts* are the elements of $C$.

Encryption and Decryption Transformation

The encryption and decryption transformation are defined below.

- Let a set $K$ be defined as the *key space* of an encryption or decryption transformation. A key $k \in K$ is an element of $K$.

- Let $E_e$ be an *encryption function* from $M$ to $C$ having an element $e \in K$ as a key. To recover a unique plaintext from each distinct ciphertext, $E_e$ must be a bijection.

- Let $D_d$ be a *decryption function* from $C$ to $M$ having an element $d \in K$ as a key.

- *Encryption of* $m$ is the process of applying a transformation $E_e$ to a message $m \in M$. 
• **Decryption** of \( c \) is the process of applying a transformation \( D_d \) to a message \( c \in C \).

• A set \( \{ E_e : e \in K \} \) of encryption functions and a corresponding set of decryption functions \( \{ D_d : d \in K \} \) having the property that for each \( e \in K \) there is a unique key \( d \in K \) such that \( D_d = E_e^{-1} \); that is, \( D_d(E_e(m)) = m \) for all \( m \in M \) determine an **encryption scheme**. An encryption scheme is sometimes referred to as a cipher.

• The key pair \((e, d)\) consists of keys \( e \) and \( d \). Note that \( e \) and \( d \) could be the same.

• A message space \( M \), a ciphertext space \( C \), a key space \( K \), a set of encryption functions \( \{ E_e : e \in K \} \), and a corresponding set of decryption functions \( \{ D_d : d \in K \} \) **construct** an encryption scheme.

**Communication Participants**

• A thing that sends, receives, or manipulates information is called communication **entity**. An entity may be a person, a computer terminal, etc.

• In a two-party communication, the entity which transmits information is called a **sender**.

• In a two-party communication, the entity which is the intended recipient of information is called a **receiver**.

• In a two-party communication, an entity which tries to intercept, manipulate (add, delete, or substitute) or impersonate the information transmitted is called an **adversary** (attacker).

**Channels**

• A **channel** is a communication path capable of transmitting data.

• A channel which is not accessible to an adversary for reordering, deletion, insertion, or reading information is a **secure channel**.

• A channel which is accessible to an adversary for reordering, deletion, insertion, or reading information is an **unsecured channel**.
Achieving Confidentiality

Confidentiality of information could be achieved in a two party communication system by secretly exchanging a key pair \((e, d)\) between the communication participants (i.e., \(e\) for the sender and \(d\) for the receiver), computing \(c = E_e(m)\), transmitting \(c\) to the receiver by the sender, and recovering the original message \(m\) at the receiver by computing \(D_d(c) = m\).

Security

In cryptography it is expected that the sets \(M, C, K, \{E_e : e \in K\}, \{D_d : d \in K\}\) are public knowledge. According to Kerckhoffs' desiderata, in a two party communication system, only the key pair \((e, d)\) must be kept secret. Additional security may be gained by hiding \(\{E_e : e \in K\}\) and \(\{D_d : d \in K\}\). The security of the entire scheme should not be based on keeping the encryption and decryption transformation secret.

Definition Without prior knowledge of the key pair \((e, d)\), if an adversary can systematically recover plaintext from corresponding ciphertext, we say the encryption scheme is breakable.

An exhaustive search of the key space is the process of trying all possible keys in the encryption process. Suppose a correct key is found, the encryption scheme is said to be breakable. To make this approach computationally infeasible, the size of the key space should be large.

Cryptology

- Cryptanalysis is the process of employing mathematical techniques in breaking the crypto-system.

- A cryptanalyst is the adversary who involves in cryptanalysis.

- Cryptology is the study of cryptography and cryptanalysis.

- A cryptosystem refers to a set of algorithms which provides security to information. This term is commonly used for encryption schemes which provide confidentiality.

Cryptographic techniques are typically divided into two generic types: symmetric-key and public-key. Encryption methods of these types will be discussed separately in the following sections.
2.2.2 Symmetric-key Encryption

**Definition** An encryption scheme with encryption functions \( \{E_e : e \in K\} \), decryption functions \( \{D_d : d \in K\} \) and a key space \( K \) is known to be symmetric if it is possible to compute \( d \) from \( e \) and vice versa for each associated key pair \( (e, d) \). In most symmetric key encryption schemes, \( e \) and \( d \) are found to be the same.

![Diagram of symmetric-key encryption](image)

**Figure 2.1:** Two-party communication using encryption, with a secure channel for key exchange. The decryption key \( d \) can be efficiently computed from the encryption key \( e \)

Figure 2.1 describes a two-party communication using symmetric-key encryption. The difficulties with this scheme are finding a method to agree upon the keys to be used and securely communicating those keys between the communication participants. In symmetric key system, both keys \( e \) and \( d \) must be kept secret as one key could be computed from the other. In Figure 2.1 only the encryption key \( e \) is transmitted since both participants can construct the decryption key \( d \) from the encryption key \( e \).

There are two class of symmetric-key encryption schemes which are commonly distinguished as block ciphers and stream ciphers. Most well-known symmetric-key encryption techniques are block ciphers. For a detail description of these ciphers, please consult [1].
2.2.3 Public-key Cryptography

**Definition:** An encryption scheme with encryption functions \( \{E_e : e \in K\} \), decryption functions \( \{D_d : d \in K\} \), and a key space \( K \) is known to be a public key encryption scheme if for each key pair \((e, d)\), key \( e \) is made public while the other key \( d \) is made private. Computation of \( d \) from \( e \) must be infeasible in order to make the scheme secure. In most asymmetric key encryption schemes, \( e \) and \( d \) are found to be different. For a detail description of public-key ciphers, please consult [1].

![Diagram of public-key cryptography](image)

Figure 2.2: Encryption using public-key techniques

2.2.4 Symmetric-key vs. Public-key Cryptography

Symmetric-key and public-key encryption have a number of complementary advantages. We summarize them here.

The advantages of symmetric-key ciphers are: Symmetric-key ciphers can be made to produce high rate of output. These ciphers have short keys. Hash functions, and computationally efficient digital signature schemes could be designed by employing symmetric key ciphers.
The disadvantages of symmetric-key cryptography are: Both keys must be secret. Management of key pairs in a large network might be difficult. Larger keys are needed by digital signature mechanisms arising from symmetric-key encryption.

The advantage of public-key cryptography are: No need to keep both the keys secret. Only the private key is kept secret. A trusted third party is needed to administer the keys on a network. No need to change keys often. Efficient digital signature schemes are designed using public-key schemes. The public key used in a verification function is smaller than the symmetric-key used in a verification function. Small number of keys are used in a large network.

The disadvantage of public-key encryption are: Most popular public-key encryption methods produce low rate output. Generally larger sized keys are used. The signatures produced by public key encryption schemes are larger than that of signatures produced by symmetric key encryption schemes. Public-key encryption schemes are not proven to be secure.

The key points to note are:

1. public-key cryptology facilitates efficient signatures (particularly non-repudiation) and key management; and

2. symmetric-key cryptography is efficient for encryption and some data integrity applications.

2.2.5 Attacks on Encryption Schemes

The hidden elements of a crypto-system are revealed by an attack. The secret key of a symmetric encryption scheme, private key of an asymmetric encryption scheme, or plain-image of a known cypher-image are examples of the hidden element.

We classify the attacks with respect to symmetric and asymmetric encryption schemes. In order to evaluate the security of symmetric-key cipher, the following assumptions are made: (1) the channel used to transmit data is accessible to an adversary, (2) the encryption scheme is a public knowledge and therefore known to an adversary, and (3) only the keys are kept secret (Kerckhoff’s assumption) and therefore not known to an adversary. The class of attacks under these assumptions are:

- Ciphertext only attack: the adversary knows ciphertexts only. The goal is to find the key or message encrypted.

- Known plaintext attack: a collection of message ciphertext pairs are known to the adversary. The aim is to determine the key or decrypt some new ciphertexts.
• Chosen plaintext attack: By choosing messages, the corresponding cyphertexts are produced. The goal of attack is same as known plaintext attack.

The goals of attack to a public-key encryption scheme are (1) the recovery of plaintext from ciphertext and (2) the recovery of private key.

The chosen-plaintext attack on a public-key encryption scheme is meaningless because the encryption scheme is public. More appropriate attack for a public-key encryption scheme is chosen-cipherattack attack.

• Chosen ciphertext attack: the adversaries can select their own cyphertexts and determine the messages that produced them. The goal is to find the key or encrypt new messages.

2.3 Authentication: Cryptographic vs. Image

In this section, we will describe cryptographic authentication and watermarking for image authentication and define basic terminology and concepts. First of all, we define what authentication is.

Data Origin Authentication Data origin authentication or message authentication techniques guarantee the identity of the entity that originates the message to an entity which receives a message.

Often messages with additional information are given to the receiver so that the receiver can determine the identity of the sender.

2.3.1 Cryptographic Authentication

Cryptographic authentication, and non-repudiation of messages are provided by digital signature schemes. The purpose of digital signature is to provide a means for an entity to bind its identity to a piece of information. Though message authentication codes (MAC) (or keyed hash functions) provide authentication, they do not guarantee non-repudiation. Digital Signatures as opposed to Message Authentication Codes guarantee non-repudiation.

Digital signature schemes involve two algorithms, one which signs an image or the digest of an image obtained by hashing, and the other that verifies the signature. Verification algorithm is accessible to all potential receivers, while the signing algorithm is used only by the signer of an image. Signing of an image involves annexing the
signature produced on the image to the image itself and verification of an image involves applying the verification algorithm to the image signature pair and a public information about the alleged signer, and then producing a "yes" or "no" answer depending on whether the message signature pair matches or not.

**Message Signature Algorithm**

- $M$ is the set of messages which can be signed.
- $S$ is a set of elements called signatures, possibly binary strings of a fixed length.
- $S_A$ is a transformation from the message set $M$ to the signature set $S$, and is called a signing transformation for entity $A$. The transformation $S_A$ is kept secret by $A$, and will be used to create signatures for messages from $M$.
- $V_A$ is a transformation from the set $M \times S$ to the set \{true, false\}. $V_A$ is called a verification transformation for A’s signature, is publicly known, and is used by other entities to verify signatures created by $A$.

**2.3.2 Watermarking for Image Authentication**

The objective is to detect modifications of an image by embedding imperceptible marks on the image itself. This is achieved with so-called "fragile watermarks" which are impaired by modifications. This differs from watermarking for copyright protection by being less robust (compression) or not robust to modifications.

Watermarking for authentication of images also involves two algorithms, one which embeds the watermark on the image, and the other that verifies the embedded watermark. Verification algorithm is accessible to all potential receivers, while the embedding algorithm is owned only by the entity that watermark the image. Watermarking of an image involves embedding the watermark on the image and verification of an image involves accepting the watermarked image and a public information about the alleged entity that watermarked the image, and then producing a "yes" or "no" answer depending on whether the recovered watermark is the same as the one which was embedded or not.

**Image Watermarked-image Algorithm**

- $I$ is the set of images which can be watermarked.
- $W$ is a set of images called watermarked images.
• $W_A$ is a transformation from the image set $I$ to the watermarked image set $W$, and is called a \textit{watermarking} for entity $A$. The transformation $W_A$ is kept secret by $A$, and will be used to create watermarked images for images from $I$.

• $V_A$ is a transformation from the set $W$ to the set $\{true, false\}$. $V_A$ is called a \textit{verification transformation} for $A$'s watermark, is publicly known, and is used by other entities to verify watermarks created by $A$.

### 2.3.3 Attacks on Cryptographic Authentication Schemes

The attack known for cryptographic authentication scheme such as a signature or a MAC is \textit{forgery}, that is, produce signatures or MACs which will be accepted as those of some other entity. The criteria for breaking a signature or a MAC follows.

1. \textit{key recovery} the key of the signer is able to be found by an adversary.

2. \textit{selective forgery} valid signatures or MACs for a class of messages are created by the adversary without the direct involvement of the legitimate signer.

3. \textit{existential forgery} valid signatures or MACs for a class of messages are created by the adversary. There is no control over the messages whose signatures have been forged. The forgery may involve the legitimate signer.

### 2.3.4 Attacks on Fragile Watermarking Schemes

An attack reveals the protected elements of a watermarking-system. These protected elements could be embedded watermark, or a watermark key.

The following are the known attacks on fragile watermarking schemes \cite{73}.

• Blind attack: arbitrarily changing the image assuming no mark is present.

• Image transfer attack: modifying the marked image without affecting the embedded mark.

• Mark transfer attack: modifying the mark without affecting the image itself.

• Removal attack: removing the mark and leaving no remnants of its existence.
2.4 JPEG Compression

The JPEG (Joint Photographic Experts Group) standard for still image compression has been recommended by ISO (International Standards Organization) and IEC (International Electro-Technical Commission). Because of the underlying block-based Discrete Cosine Transform (DCT), the performance of these coders deteriorates at low bit rates. Most recently, the wavelet transform has become popular within the field of image compression because of its superior performance at higher compression ratios and many sophisticated wavelet-based schemes for compression have been invented.

This section presents an overview of the DCT-based JPEG coding, the goal of new wavelet based JPEG-2000 standard, and the relationship between filtering (low-pass and high-pass) and JPEG compression.

2.4.1 JPEG: DCT-Based Image Coding

Figure 2.3 shows the key processing steps which are the heart of the DCT-based modes of operation. This figure illustrates the special case of single-component (grayscale) image compression.

8x8 FDCT and IDCT

Encoder divides an image into 8x8 blocks, transforms pixel representations from unsigned integers in the range [0, 2^P - 1] to signed integers in the range [-2^(P-1), 2^(P-1) - 1], and applies Forward DCT (FDCT). Decoder, after applying Inverse DCT (IDCT), outputs 8x8 sample blocks to form the reconstructed image. The following equations are the mathematical definitions of the 8x8 FDCT and 8x8 IDCT.

\[ F(u, v) = (1/4)C(u)C(v)\sum_{x=0}^{7} \sum_{y=0}^{7} f(x, y) \cdot \cos(2x + 1)u\pi/16 \cdot \cos(2y + 1)v\pi/16 \]

\[ f(x, y) = (1/4)\left[ \sum_{x=0}^{7} \sum_{y=0}^{7} C(u)C(v)F(u, v) \cdot \cos(2x + 1)u\pi/16 \cdot \cos(2y + 1)v\pi/16 \right] \]

Where: \( C(u), C(v) = 1/\sqrt{2} \) for \( u, v = 0 \);

\( C(u), C(v) = 1 \) otherwise.

DCT and Discrete Fourier Transform (DFT) are related to each other. FDCT and IDCT used in the DCT based JPEG compression can be viewed as harmonic analyser and harmonic synthesizer, respectively. The FDCT takes each 8x8 block of source image as input signal and decomposes it into 64 orthogonal basis signals. Output of
FDCT contains the 64 "spatial frequencies" which show the input signal’s "spectrum". They are also called the set of 64 basis-signal amplitudes or "DCT coefficients"

The DCT coefficients are also considered as the relative amount of the 2D spatial frequencies contained in the 64 point input signal. "DC coefficient" is the coefficient with zero frequency and "AC coefficients" are the remaining 63 coefficients. The basis for data compression is achieved by concentrating most of the signals in the lower spatial frequencies during FDCT. For a $8\times8$ sample block most of the high frequencies are zero and are not encoded.

IDCT is the reverse operation of FDCT at the decoder. It involves accepting 64 DCT coefficients and reconstructing 64 point output image signal by summing the basis signals. DCT can be regarded as a one to one mapping from the image to the frequency domain. DCT introduces no loss to the source image unless inaccuracies are caused by computations or quantization; it merely transforms the source image into a
frequency domain.

**Quantization**

Output of FDCT (64 DCT coefficients) are uniformly quantized using the quantization table specified at the input. The step sizes of the quantizer, which take values in the range of 0-255, are the elements of the table. The goal of quantization is to gain compression by removing DCT coefficients which are not perceptually significant. Since quantization is a many to one mapping, it is regarded as lossy and it is the principal source of lossiness in DCT-based encoder.

The division of each DCT coefficient by its corresponding quantizer step size, and rounding the result of the division to the nearest integer define quantization.

\[ F^Q(u, v) = \text{Integer Rounding}(F(u, v)/Q(u, v)) \]

Dequantization is the inverse of quantization. It is defined as multiplication of a quantized value by its corresponding quantizer step size. The results are the representation appropriate for input to the IDCT:

\[ F^{Q'}(u, v) = F^Q(u, v) * Q(u, v) \]

The goal of compression is to compress the image to its maximum extent without noticeable visual artifacts. To accomplish this, each step size should be chosen in such a way that the visual contribution of its corresponding basis function is within ”just noticeable difference”. The quantization step sizes are also functions of image characteristics, display characteristics, and viewing distance. Psychovisual experiments can be performed to determine the best quantization step sizes on images which have these characteristics well defined.

**DC Coding and Zig-Zag Sequence**

DC coefficients are encoded differently from the AC coefficients. Since DC coefficients are measures of the average value of the 64 image pixels, the correlation between the DC coefficients of adjacent 8x8 blocks is very high. Because of this reason, quantized DC coefficient is encoded as the difference from the DC term of the previous block in the encoding order as shown in Figure 2.4. This special encoding strategy is useful, as DC coefficients consist of a significant portion of the total energy.
Finally, the "zig zag" ordering of quantized coefficients is carried out, as shown in Figure 2.4. This ordering helps entropy coding achieve higher compression by placing low-frequency (mostly non-zero) coefficients before high-frequency (mostly zero) coefficients.

**Entropy Coding**

The last step in DCT based encoding is entropy coding. In this step, an additional compression is gained losslessly by encoding the quantized coefficients based on statistical frequency distributions. Huffman coding and Arithmetic coding are specified by JPEG proposals.

Entropy coding can be considered as a two step process. In the first step an intermediate sequence of symbols are produced from the zig-zag ordered quantized coefficients. In the next step, those symbols are converted into a binary stream which has no detectable boundaries. The DCT based mode of operation and the entropy coding method determine the definition of the intermediate symbols.

If Huffman coding is chosen by the application, one or more sets of Huffman code tables must be specified. For a specific image, the tables used to compress must be the same as the tables used to decompress. A predefined Huffman table may be used as a default or a table computed based on the statistical property of a given image may be used.

On the contrary, if arithmetic coding is chosen by the application, JPEG needs no tables to be specified externally. This is because the arithmetic coding specified in
the JPEG proposal adapts to the image statistics as it encodes the image. For many images, arithmetic coding has produced 5 – 10% better compression than Huffman. However, for high speed hardware implementation, arithmetic coding has been felt more complex than Huffman coding.

Compression and Picture Quality

For color images with moderately complex scenes, DCT based modes of operation typically produce the following levels of picture quality for the indicated ranges of compression. These levels are only a guideline - quality and compression can vary significantly according to source image characteristics and scene content.

- 0.25-0.5 bits/pixel: moderate to good quality, sufficient for some applications;
- 0.5-0.75 bits/pixel: good to very good quality, sufficient for many applications;
- 0.75-1.5 bits/pixel: excellent quality, sufficient for most applications;
- 1.5-2.0 bits/pixel: usually indistinguishable from the original, sufficient for the most demanding applications.

2.4.2 JPEG-2000: Image Compression Standard

The new standard, which is also known as JPEG-2000 [3, 4] includes improved low bit-rate compression performance, lossless and lossy compression, continuous-tone and bi-level compression, visual weighting, scalar quantization, region of interest, and error resilience. The input image is first transformed using a wavelet transform. Integer, fixed point, or floating point wavelet transform can be used. Different wavelets are also allowed to be used. The transform coefficients are then quantized using either a scalar or trellis coded quantizer. The transform coefficients can be grouped together before quantization by classifying them into two groups. This allows the quantizer to adapt to the image characteristics at different region of the image. The quantized coefficients are then coded using bit-plane, context arithmetic coder.

2.4.3 Compression vs. Filtering

Images comprise of both smooth and busy regions. The amount of busyness/smoothness determines the level of activity in an image. An image is said to have a higher level of activity if the amount of busyness exceeds the amount of smoothness, and an image is said to have a low level of activity if the amount of smoothness exceeds the amount
of busyness. An image with higher level of activity will have more high frequency components than low frequency components, and an image with low level of activity will have more low frequency components than high frequency components.

Human perception is sensitive to changes in low frequency components than to changes in higher frequency components of an image. JPEG compression exploits these properties by removing the high frequency components while keeping the low frequency components in tact. This is accomplished by employing a proper quantization technique on the transformed coefficients. As commonly said in signal processing, JPEG compression could be described by filtering techniques such as, low-pass or high-pass filtering. The term low-pass refers to removing the high frequency components of a signal, while keeping the low frequency components in tact, and the term high-pass refers to removing the low frequency components of a signal, while keeping the high frequency components in tact. This clearly shows JPEG compression is a low-pass filtering.

2.5 Combined Compression and Crypto Systems

This section describes the need for combining compression with crypto systems.

2.5.1 Compression Surviving Authentication

Two fundamental components of compression are redundancy reduction and irrelevancy reduction. Redundancy reduction aims at removing duplication from the signal source. It also refers to the removal of statistical redundancy. From this type of reduction, the information that has been reduced can be recovered. Irrelevancy reduction aims at removing parts of the signal that will not be noticed by the signal receiver. It also refers to the removal of spectral (frequency) redundancy. From this type of reduction, the information that has been lost can not be recovered.

Since message compression uses redundancy reduction (ie. the removal of statistical redundancy), it is lossless and there is no question of compression surviving authentication. This is because when a compressed stream is decompressed the original message will be replicated without any loss of information.

Lossy compression scheme such as JPEG, uses both redundancy reduction and irrelevancy reduction. Since irrelevancy reduction (ie. the removal of frequency redundancy) is involved in JPEG compression, although the image impression is preserved after compression and decompression, some amount of information is lost. Even in
this situation compressed images have to be some how authenticated. This is what we mean by \textit{compression surviving authentication}. This could be possible, only if the authentication system is designed based on some features of an image, that survive after certain level of information loss due to compression.

\subsection{Combined Compression and Encryption}

Compression before encryption increases security by removing the redundancy. Ciphertexts produced by encryption schemes are random sequences with less redundancy. Less redundant sequences such as ciphertexts can not be compressed much. Hence compression before encryption is much more efficient than encryption before compression. Combining compression and encryption has the advantage of efficiency in terms of complexity, as long as the compression performance is not sacrificed very much.

\section{Conclusion}

In this chapter, we have introduced the basic definition of cryptographic and image authentication, symmetric and asymmetric encryption, and JPEG compression. The possible attacks to these schemes and the advantages of combining secrecy with compression have also been described.
Chapter 3

Cryptographic Hash Functions and Data Integrity

3.1 Introduction

In cryptography, hash functions play a fundamental role. A precise definition of a hash function follows before further discussions.

**Definition** A hash function $h$ is a computationally efficient function from a domain $D$ to an image $R$, mapping a message bit-strings of arbitrary finite length to a binary string of fixed length $n$ bits, called hash value.

Since a hash function $h : D \to R$ with $|D| > |R|$ is a many to one mapping, collusion (pair of inputs with identical output) is unavoidable. Hash value produced by a hash function is the message digest, which compactly represents the image of a message. In cryptography, it is used with the message string to uniquely identify the message. A function to qualify as a cryptographic hash function, finding two distinct inputs $x$ and $y$ that hash to a common value, and determining a preimage $x$ for a specific hash value $y$ such that $h(x) = y$, must be difficult (or computationally infeasible).

Data integrity assurances are given by signature schemes which sign the hash value of a message. Here, hash value of a message is considered as a compact representative of the message. The receiver of a message signature pair, computes the hash value of the message, and determines if the received signature is the same as the one computed on the hash value. This process is also known as verification of signature.

Hash functions defined above are public knowledge. These functions do not take additional inputs as secret keys. They are also called modification detection codes (MDCs) when the usage is for detecting manipulation of messages. Another class of functions related to hash functions are message authentication codes (MACs). MACs use symmetric techniques involving a secret key in providing authentication. They are functions with two distinct inputs, a message and a secret key. The output produced by MACs are of a fixed length (say $n$ bits). Collusion freeness property also holds for MACs.
That is, for a valid key, two distinct messages should not produce the same output. MACs can provide data integrity and symmetric data origin authentication simultaneously. For a more detail description of hash functions and MACs, please consult [1].

3.2 Usage of Hash Functions

The usage of hash functions can be explained as follows. A hash value, computed on a message $x$, is digitally signed to protect the integrity. At a subsequent time, hash value computed on the received message is compared against the protected hash value of the original message: In case, they are equal the original message has not been altered.

By hashing a message, the problem of preserving the integrity of a large message is reduced to a small fixed size hash-value. The existence of collusion is unavoidable because of the many to one property of hash functions. The unique association between a hash value and its input is, at best, be provided in the computational sense. In practice, hash value should be uniquely identifiable with a single input, and collusion must be computationally difficult to find.

3.3 General Classification

Hash functions can be divided into two classes: unkeyed hash functions, which accept a single message as input parameter; and keyed hash functions, which accept a single message and a secret key as two input parameters. To help discussion, a hash function (unkeyed) and a MAC (keyed) are defined as follows.

**Definition of hash function** A hash function is a function $h$ which has, as a minimum, the following two properties:

1. compression: $h$ maps an input $x$ of arbitrary finite bitlength, to an output $h(x)$ of fixed bitlength $n$.

2. ease of computation: given $h$ and an input $x$, $h(x)$ is easy to compute.

**Definition of Message Authentication Codes** A MAC is a function $h$ which has, as a minimum, the following two properties:

1. compression: $h$ maps input $x$ of arbitrary finite bitlength and a secret key $k$, to an output $h(x, k)$ of fixed bitlength $n$. 
2. *ease of computation:* given $h$, input $x$ and a secret key $k$, $h(x, k)$ is easy to compute.

As defined here, *hash function* implies an unkeyed hash function and MAC implies a keyed hash function.

For practicality, a goal-oriented classification of these functions are given below:

1. **Hash functions** (MDCs)
   The aim of hash functions (*Manipulation Detection Codes* (MDCs) or *Modification Detection Codes* (MDCs), or *Message Integrity Codes* (MICs)) is to provide a representative image or *hash* of a message. The additional properties needed to satisfy are given below. The ultimate goal of hash functions is to provide data integrity with additional mechanisms. These hash functions may be further classified as follows;

   (a) **One-way Hash Functions** (OWHFs): for these functions, determining an input which hashes to a specified hash-value is computationally hard;

   (b) **Collision Resistant Hash Functions** (CRHFs); for these functions, determining two inputs which hash to the same hash-value is computationally hard.

   The relationship between these two are given in the next section.

2. **Message Authentication Codes** (MACs)

   Providing data integrity and data origin assurances, without the use of any additional mechanisms, is the ultimate goal of MACs. MACs take two distinct input parameters, a message input and a secret key; They are also known as *keyed* hash functions.

   According to Kerckhoff’s assumption, algorithmic specification of a hash function is public knowledge. That is, knowing the input message, any one can compute the hash value. In the case of MACs, any one knowing the input message and the secret key can compute the hash result.

### 3.4 Basic Properties and Definitions

Three main properties of a hash (unkeyed) function $h$ with inputs $x$, $x'$ and outputs $y$, $y'$ are listed below. These properties are given in addition to ease of computation and compression.
1. **Preimage resistance:** for a given output $y$, it is computationally infeasible to find any input $x'$ which hashes to that output, i.e., to determine any preimage $x'$ such that $h(x') = y$ is impossible.

2. **2nd-preimage resistance:** for a given output $y$ with an input $x$, it is computationally infeasible to find another input $x'$ that hash to the same output $y$, i.e., to determine a 2nd-preimage $x' \neq x$, such that $h(x) = h(x')$ is hard.

3. **Collision resistance:** it is computationally infeasible to find any two distinct inputs $x, x'$ which hash to the same output, i.e., to determine two distinct inputs $x$ and $x'$ such that $h(x) = h(x')$ is difficult.

One motivation for above properties follows. If a digital signature is applied to the hash value $h(x)$ of a message $x$, and if $h$ is not 2nd pre-image resistant, an adversary may find another $x'$ such that $h(x) = h(x')$ and claim $x'$ has been signed instead of $x$. In case the message $x$ is known to the adversary, only a collision pair $(x, x')$ needs to be found rather than the difficult task of finding a second preimage of $x$; here collision resistance is also necessary.

**One-way hash function (OWHF):** is a hash function $h$ offering ease of computation and compression with the properties of preimage resistance and 2nd-preimage resistance.

**Collision resistant hash function (CRHF):** is a hash function $h$ offering ease of computation and compression with the properties of 2nd-preimage resistance and collision resistance.

**Message authentication codes (MAC):** are family of functions $h_k$ with $k$ as a secret key satisfying the following properties:

1. **Ease of computation:** $h_k(x)$ is easy to compute for a given function $h_k(x)$, input $x$, and a secret key $k$. The result is known as a *MAC-value* or a *MAC*.

2. **Compression:** for an arbitrary input $x$ of finite bit length, output of $h_k(x)$ is of fixed bit length $n$.

Furthermore, for a known function family $h_k(x)$ ($k$ unknown to an adversary), the property given below holds:
3. **Computation-resistance:** from a known set of text-MAC pairs \((x_i, h_k(x_i))\), a new text-MAC pair \((x, h_k(x))\) for any input \(x \neq x_i\), is computationally infeasible to compute. (including possibly for \(h_k(x) = h_k(x_i)\) for some \(i\)).

Computation resistance is an important property for MAC algorithms to hold in order to avoid MAC forgeries. Computation resistance implies key non-recovery. That is, the recovery of key \(k\) must be computationally impossible, for a given set of text MAC pairs \((x_i, h_k(x_i))\). Non-recovery of key does not imply computation-resistance. That is, a key does not always be known to forge a new MAC.

When a key is known, **MAC resistance** does not say whether a MAC has to be preimage and collision resistant.

### 3.4.1 Attacks on Hash functions and MACs

1. **Objectives of adversaries in hash functions**

   The goal of an adversary who wants to ”attack” a hash function is as follows:

   - Attack a OWHF: for a known hash-value \(y\), determine a preimage \(x\) such that \(y = h(x)\); or given one such pair \((x, h(x))\), determine a second preimage \(x'\) such that \(h(x') = h(x)\).

   - Attack a CRHF: determine two inputs \(x, x'\) such that \(h(x') = h(x)\).

   A CRHF must be constructed to withstand standard birthday attacks.

2. **Objectives of adversaries in MACs**

   The goal of an adversary for a MAC algorithm is as follows:

   - Attack a MAC: given one or more message-MAC pairs \((x_i, h_k(x_i))\), without knowing \(k\), determine a new text-MAC pair \((x, h_k(x))\) for some message \(x \neq x_i\).

3. **Types of forgery**

   When a MAC is forged, the amount of damage done may differ based on the level of control an adversary has over the value of \(x\) for which a MAC may be forged. This level is differentiated as follows:

   - **Selective forgery** attacks where an adversary manages to find a new message-MAC pair for a message of his own choice.

   - **Existential forgery** attacks where an adversary manages to find a new message-MAC pair without any control over the value of the message.
Most severe attack of MAC is key recovery because it allows selective forgery. MAC forgery permits forged messages accepted as authentic. The damages may be severe even in the existential case. Similar to MACs, attacks on MDC schemes may be classified as selective or existential.

3.5 Relationship between Properties

In this section several relationships between the hash function properties stated in the preceding section are examined.

Collision resistance implies 2nd-preimage resistance of hash functions

In case $h$ satisfies collision resistance property, fix an input $x_j$. If $h$ does not satisfy 2nd preimage resistance, then it is possible to determine a distinct input $x_i$ such that $h(x_i) = h(x_j)$. That is, $(x_i, x_j)$ is a pair of distinct inputs hashing to the same output. This contradicts the collision resistance property.

One way vs. preimage and 2nd-preimage resistant

Although the term "one-way" generally means preimage resistant (computationally non-invertible), it is sometimes used to mean that a function is 2nd preimage resistant. This causes ambiguity as to whether 2nd preimage resistance guarantees preimage resistance or preimage resistance guarantees 2nd preimage resistance.

Collision resistance does not guarantee preimage resistance

Suppose $g$ be a hash function which is collision resistant and maps arbitrary-length inputs to $n$-bit outputs. Define a function $h$ as $h(x) = 1||x$ if $x$ has bitlength $n$ and $h(x) = 0||g(x)$, otherwise.

Then $h$ is an $(n + 1)$-bit hash function which is collision resistant but not preimage resistant. The identity function on fixed length inputs is collision and 2nd preimage resistant but not preimage resistant. For most CRHFs in practice, collision resistance implies preimage resistance.
3.6 Other Hash Function Properties and Applications

Additional properties of one-way hash functions needed for some applications are follows:

1. **Non-correlation:**
   Input bits and output bits must not be correlated. An avalanche property similar to that of good block cipher is desirable whereby every input bit affects every output bit.

2. **Near-collision resistance:**
   It must be hard to determine two inputs $x, x'$ such that $h(x)$ and $h(x')$ differ in only a small number of bits.

3. **Partial-preimage resistance or local one-wayness:**
   It must be as hard to recover any substring as to recover the entire input. Moreover, even if partial input is known, it must be hard to determine the rest.

3.7 Data Integrity

In this section, data integrity, data origin authentication (message authentication) and transaction authentication are discussed. In many applications, assurances of source of origin of data and data integrity are typically required. These issues can not be separated. That is, source of origin of data and integrity of data are linked with one another; data which has been altered most probably has a new source. Integrity mechanisms thus provide data origin authentication, and vice versa.

3.7.1 Data Integrity

Data integrity provides assurance that data has not been altered in an unauthorized manner since the time it was created, transmitted, or stored by an authorized source.

Requirement of data integrity verification dictates that only a subset of all candidate data items satisfies particular criteria distinguishing the acceptable from the unacceptable. Criteria that recognize data integrity includes appropriate redundancy or expectation with respect to format. Cryptographic techniques for data integrity depends on either secret information or authentic channels.
The focus of data integrity is on the bitwise composition of data. Invalidation of data integrity may occur due to insertion of bits, deletion of bits, re-ordering of bits or groups of bits, inversion or substitution of bits or any combination of these.

### 3.7.2 Data Origin Authentication (Message Authentication)

Message authentication and data origin authentication are terms similar to each other. They provide data origin authentication with respect to source of origin of message and data integrity (no uniqueness and timeliness guarantee).

Methods for ensuring data origin authentication are as follows:

1. message authentication codes (MACs)

2. digital signature schemes

3. annexing a secret authentication value to a message before encryption.

Message Authentication Codes (MACs), which provide data origin authentication based on shared secret keys, do not allow a distinction to be made between the parties sharing the keys. Therefore, they (as opposed to digital signatures) do not provide non-repudiation of data origin - either party can equally originate a message using the shared key. In case a resolution of disputes is required, a trusted third party is needed. Another alternative is to employ an asymmetric technique.

Although MACs and digital signatures are used for providing data origin assurances (authentication), they do not provide uniqueness or timeliness guarantee. Re-use or replay of message can not be detected by these techniques. Such message authentication techniques may, however, be augmented to provide these guarantees, as next discussed.

### 3.7.3 Transaction Authentication

Messages, which are to be authenticated, may be augmented with additional information to provide uniqueness and timeliness guarantee. This is called Transaction Authentication.

Appropriate use of time-variant parameters (TVPs) provides uniqueness and timeliness guarantees. These include random numbers in challenge-response protocols, sequence numbers, and timestamps. View this as a combination of message authentication and TVPs.
3.7.4 Models for Providing Data Integrity

In this section the three methods for providing data integrity using hash functions are given. The first method assumes a MAC produced on the message is attached to the message and message MAC pair is send via an insecure channel (Data integrity using a MAC alone). The second one assumes a hash produced on the message is attached to the message and encrypted message MDC pair is send via an insecure channel. (Data integrity using encryption and a hash) The last one assumes while the MDC produced on the message is sent via an authentic channel, the message is sent via an unsecured channel. (Data integrity using a hash and an authentic channel)

Data Integrity Using a MAC Alone

Providing data integrity (not privacy) is the aim of designing Message Authentication Codes (MACs). The sender of a message $x$ computes a MAC $h_k(x)$ over the message using a secret MAC key $k$ known to the intended receiver. He sends message-MAC pair(effectively $x||h_k(x)$) to the receiver. The receiver separates the received MAC from the received message, computes a MAC using the shared key, and compares the computed MAC to the received MAC. In case the MAC values agree, received message is is interpreted by the receiver as authentic - that is, it originated from the other party which knows the shared key, and has not been tampered with in transit. This corresponds to Figure 3.1(a)

Data Integrity Using Encryption and a Hash

If both confidentiality and integrity are required, then the following data integrity technique may be used. The sender of the message $x$ computes a hash value $H = h(x)$ of the message, attaches it to the data, and encrypts the augmented message using a symmetric encryption algorithm $E$ with shared key $k$, producing cipher $C = E_k(x||h(x))$. This is sent to a receiver who decrypts the message, separates the recovered data $x'$ from the recovered hash $H'$, computes the hash $h(x')$ of the received data $x'$, and compares this to the recovered hash $H'$. Suppose they agree, the recovered data is accepted as both being authentic and having integrity. This corresponds to Figure 3.1(b)

Here, encryption protects the appended hash, and message. It will be impossible for an attacker without the encryption key to alter the message without modifying the relationship between the decrypted plaintext and the recovered hash.
Data Integrity using a Hash and an Authentic Channel

In order to provide data integrity, the use of a secret key is not essential. It could be eliminated by protecting the hash by sending it via an authentic channel. The sender computes a hash of the message, transmits the message to a receiver over an unsecured channel, and transmits the hash over an independent secure channel. The receiver hashes the received message, and compares the hash to that received. Suppose they agree, the receiver accepts the data as having integrity or authentic. This corresponds to Figure 3.1(c)
3.8 Image Hashing for Authentication

Image authentication plays an important role in security and communication. Images are being transferred over the Internet and are readily available for access from any part of the world and without introducing authentication mechanism, it is almost impossible to distinguish if an image is original or being manipulated.

Using cryptographic methods to authenticate image data will result in an unworkable systems or unacceptable systems because data authentication are sensitive to single bit change in the original data while image authentication systems need to be mainly
content sensitive. This is because images undergo a range of processing including lossy compression that result in changes in bits that are deemed acceptable. Such changes must be tolerable by the authentication system while it is essential for the system to remain sensitive to malicious manipulations.

The three main approaches to image authentication have been *watermarking* in which a fragile watermark is embedded in the image, *image hashing*, also called *feature extraction*, in which a digest of the image capturing its main features is generated, and *message authentication codes* (MAC), also known as keyed hashing which uses a key in hash generation process. The image hash can be encrypted or digitally signed to generate an *authentication tag* to be appended to the image. In all approaches there is also a verification system that takes an image with the authentication information including the secret key, and produces a *true/false* value, depending on the authenticity, of the image. In recent years there has been numerous proposals in each category and the assessment of security in all cases have been ad-hoc and mainly through experiments.

Efficiency of a hash or MAC based authentication system depends on the size of the key, and the amount of computation required for generation and verification of the authentication tag.

### 3.9 Image Hashing

An image hashing algorithm $H$ takes an arbitrary image $I$ and produces a bit string $x_I$. Unlike cryptographic hash functions, image hashing algorithms must be *bit insensitive* and produce the same value for *similar images* that are defined through a set $A$ of *admissible transformations*. These are transformations that do not modify the main features of the image. We say an image $I'$ is *similar to* $I$ if $I' = a(I), a \in A$. We also need to define the set $A'$ of *inadmissible transformations*, that consists of transformations that are applied to the whole image, such as compression at a low quality factor with the aim of damaging important details of the image, and *localized modification* of the content.

The hashing algorithm must produce 'close' values for images that are obtained through acceptable transformations, and 'distant' values for images that are obtained through forbidden ones. To quantify the notions 'close' and 'distant' we require a distance function.

Some possible measures of distance between two hash values, $v$ and $v'$, are the correlation coefficient $d_{CC}(v, v')$.
\[ d_{CC}(v, v') = \frac{\sum_{i=0}^{n}(v_i - \overline{v_i})(v'_i - \overline{v'_i})}{\sqrt{\sum_{i=0}^{n}(v_i - \overline{v_i})^2} \cdot \sqrt{\sum_{i=0}^{n}(v'_i - \overline{v'_i})^2}} \]

and Euclidean distance \( d_{ED}(v, v') \).

\[ d_{ED}(v, v') = \sqrt{\sum_{i=0}^{n}(v_i - v'_i)^2} \]

If two hash values are close, that is

\[ d_{CC}(H(I), H(I')) \in (1 - \epsilon, 1 + \epsilon) \]

or

\[ d_{ED}(H(I), H(I')) < \epsilon \]

, we say the two images are similar, or collide, and attach a confidence level to this statement. The threshold and the confidence level needs to be determined using the set \( A \) and \( A' \).

Image hash algorithms that are used for image authentication must have the property that it must be difficult to construct two images \( I \) and \( I' \) that collide and \( I \neq a(I') \), \( a \in A \). This property is also called collision intractability \cite{1} and ensures that a manipulated image has a different hash value and so the authentication tag is only valid for similar images.

A hash algorithm may produce a close value for an image that is obtained through a non-admissible transformation, that is a collision, or produce a distant value for an image obtained through admissible transformation. These two will result in a false positive and false negative result, when the hashing is used for image authentication. For a good hash algorithm, the probability of false positive and false negative, taken over all possible inputs must remain small.

### 3.10 Previous Work

The early proposals for authentication of digital images produce a signature by encrypting a hash version of the image while the digital camera produces it \cite{31} or embed some secret code or watermark to the image \cite{32}.

The authentication systems based on hash or MAC of an image proposed so far rely on some kind of feature extraction mechanism from the image. Most of the hash based systems uses edge detection for feature extraction \cite{34, 35} while the others use histograms of image blocks \cite{33} or responses of a wavelet transform \cite{36, 37}. An early proposal for a MAC uses the invariant property of the relationship between DCT
coefficients, and a recent one [45] uses the fact that the highest order bit of each pixel is least affected in low-pass filtering.

The other methods that have been proposed are based on the Public Key Encryption [33, 34]. A private key is used to encrypt a hashed version of the image to produce the signature and annexed to the image for verification. The authentication process of the image needs an associated public key to decrypt the signature.

These methods also give some provision for authenticating images that have undergone acceptable level of modifications such as JPEG compression, filtering for noise removal or smoothing, while rejecting any other malicious manipulations such as removal of an object.

3.10.1 A Robust Image Authentication Method Distinguishing JPEG Compression from Malicious Manipulation- By Ching-Yung Lin and Shih-Fu Chang

This method of authentication [38, 39, 40] is based on the invariance property of the relationship between DCT co-efficients of the same positions in separate blocks of an image. It has been proved that this relationship will be preserved even if these coefficients are quantized in a JPEG compression process regard less of how high the compression ratio is.

Moreover, their scheme generates a signature, which is an encrypted form of the feature codes or hash of the image and it is stored separately. When the user needs to verify the authenticity of the image he receives, he should decrypt this signature to obtain the feature codes of the image that has been sent, compute the feature codes from the received image, and compare these two. If they match, this image can be claimed to be authentic.

Signature Generation

Each $8 \times 8$ block of an image is transformed to the DCT coefficients. The feature codes are generated according to two controllable parameters: mapping function, $W$, and selected positions, $b$, in the DCT domain. Given a block $p$ in an image, the mapping function is used for selecting the other block to form a block pair, i.e., $q = W(p)$. A coefficient position set, $b$, is used to indicate which positions in a $8 \times 8$ block are selected. The feature codes of the image records the relationship of the difference value, $Fp(i) - Fq(i)$, and zero at $N$ positions of $b$. If the difference is larger than or equal to $T_i$, a bit 1 is represented; otherwise, a bit 0 is recorded. This process is applied to all
blocks to ensure the whole image is protected. In the last step, the feature codes are encrypted with a private key by using the Public Key Encryption method.

**Signature Verification**

Given a signature derived from the original image and a JPEG compressed image bitstream, $Bm$, for authentication, at the first step, we have to decrypt the signature and reconstruct DCT coefficients from $Bm$. Because the feature codes decrypted from the signature record the relationship of the difference values and $T_i$, they indicate the sign of the difference between the difference of DCT coefficients and $T_i$, despite the changes of the coefficients incurred by lossy JPEG compression. If these constrains are not satisfied, this image has been manipulated by another method.

**Size of MAC**

The size of MAC is equal to $(b_1+b_2+...+b_N)*(p/2)$+size of $T_1+...+size of T_N$+size of the seed of the mapping function, where $b_i$ are the number of coefficients starting from DCT co-efficient in a zig-zag order compared in each pair of blocks using the corresponding threshold $T_i$, $p$ is the number of blocks of size $8 \times 8$ used in DCT transform.

The size of the MAC can be changed by choosing different values for $b_1,...,b_N$.

**Computation**

$(b_1 + b_2 + ... + b_N) * (p/2)$ comparisons are needed.

**Size of Key**

The size of key is the sum of the size of the key used in the chosen encryption scheme, the size of the seed used in the mapping function $W$, and the size of $b_1,...,b_N / T_1,...,T_N$ if hidden as keys.

**Probability of Success in Cheating**

The probability of success in cheating depends on finding the keys mentioned above. More precisely, if one can choose the right pairing without knowing the seed, correct MAC can be computed. Thus, the probability of success in cheating relies on the probability of success in finding the right mapping which is also equal to $1/(pCp/2 \times p/2!)$.

If the feature code computation is public except for the seed and the mapping function used, the scheme is easily vulnerable to attacks as mentioned above.
3.10.2 Towards Robust Content Based Technique for Image Authentication by Maria Paula Queluz

In Queluz's method [34] gradient computed at pixel position with the Sobel operator is compared with a threshold histogram to obtain edge patterns. Canny's edge detector for edge extraction is also suggested. Her scheme sub-samples edge image by employing a majority filter technique proposed in JBIG and then entropy codes (arithmetic coding - modified READ or JBIG) the edge bit map. During the authentication process confidence measures are computed on each edge error pixel to determine the authenticity of the received image.

More-over, this scheme also generates a signature, which is an encrypted form of the entropy coded edge bit map and it is stored separately. When the user needs to verify the authenticity of the image he receives, he should decrypt this signature to obtain the entropy coded edge bit map of the image that has been sent, compute the entropy coded edge bit map of the received image, and compare these two. If they match, this image can be claimed to be authentic.

Size of Hash

The size of hash is equal to the storage needed to keep the final range of the output of arithmetic coder plus the storage needed to store the frequency table. More precisely, it can be given by $(2*p) + m*(q+2*p)$, where $p$ is the bytes needed to store a real number, $m$ is the number of distinct symbols in the edge bit map, and $q$ is the bytes needed to store each symbol.

Computation

The total computation involved is equal to the sum of the operations in edge detection, operations in edge compression, and the operations in entropy coding.

The total number of operations in edge detection is given by $(18*n)*2+3$ additions and multiplications, and $n$ comparisons, assuming a $3x3$ sobel mask is used and the number of pixels is $n$. The total number of operations in edge removal using a $3x3$ majority filter involves $9$ comparisons per pixel, that is, $9*n$ comparisons altogether. The entropy coding involves $n$ comparisons and $n$ additions for constructing the frequency table, and a multiple of $n$ operations for encoding the symbols, where $n$ is the number of pixels in an image. (Computation $O(n)$)
Size of Key

Since none of the parameters are hidden as keys in edge detection, edge compression, or entropy coding, the size of the key is equal to the size of the key of the encryption scheme used.

Probability of Success in Cheating

Because the scheme is edge dependent, it will be very hard to construct another image with exactly the same edge impression having a different meaning. Colour images can be manipulated by changing the colour without modifying the edges.

The probability of success in finding another image with the same hash is one (100%), and it can be illustrated as follows,

Gradients computed on two images in Table 3.2 using the Sobel masks in Table 3.1 are given in Table 3.3.

\[
\begin{array}{c c c c c}
-1 & 0 & 1 & 1 & 2 & 1 \\
-2 & 0 & 2 & 0 & 0 & 0 \\
-1 & 0 & 1 & -1 & -2 & -1 \\
\end{array}
\]

Table 3.1: Sobal masks

\[
\begin{array}{c c c c c c c c c c c c c c c c c c}
5 & 1 & 4 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 5 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 5 & 1 & 1 & 1 & 1 & 1 & 9 & 1 & 1 & 1 \\
1 & 1 & 1 & 5 & 1 & 1 & 1 & 9 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 5 & 1 & 1 & 1 & 1 & 9 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 5 & 1 & 1 & 1 & 1 & 9 & 1 \\
\end{array}
\]

Table 3.2: Two images having a digonal edge with different intensities

\[
\begin{array}{c c c c c c c c c c c c c c c c c c}
0 & -8 & -8 & 0 & 0 & -16 & -8 & 0 \\
8 & 0 & -8 & -4 & 16 & 0 & -16 & -8 \\
4 & 8 & 0 & -8 & 8 & 16 & 0 & -16 \\
0 & 4 & 8 & 0 & 0 & 8 & 16 & 0 \\
\end{array}
\]

Table 3.3: Gradients computed
If a threshold of 4 is used, same edge images will be produced on the above two distinct images.

### 3.10.3 Compression Tolerant Image Authentication by Sushil Bhattacharjee and Martin Kutter

The hashing scheme proposed by Bhattacharjee [36] is also tolerant to lossy image compression but can detect malicious image manipulations. Their scheme relies on visually salient image features which are extracted using scale interaction based on Mexican-Hat wavelets. They also claim the results so far are promising.

The process of image authentication consists of two parts: generation of the digital signature of the original image, and subsequent verification of the digital signature of the image to be authenticated with the stored signature of the original.

#### Digital Signature Generation

The scheme for generating the digital signature of an image involves two steps. First, the co-ordinates of visually important image structure are extracted. These co-ordinates give the feature points on which the authentication scheme is based. The steps involved in detecting feature points are as follows:

1. Define the feature detection function $P_{i,j}$ as: $P_{i,j}(x) = |M_i(x) - \gamma M_j(x)|$, where $M_i(x)$ and $M_j(x)$ represent the responses of Mexican Hat wavelet at image location $x$ for scales $i$ and $j$ and $\gamma = 2^{-(i-j)}$.

2. Determine points of local maxima of $P_{i,j}(x)$. These maxima correspond to the set of potential feature points. A circular neighborhood with a radius of 5 pixels has been used to determine the local maxima of $P_{i,j}(x)$.

3. Accept a point of local maxima in $P_{i,j}(x)$ as a feature point if the variance of the image pixels in the neighborhood of the points is higher than a threshold. A $7 \times 7$ neighborhood around the point has been used for computing the local variance. A candidate point is accepted as a feature point if the corresponding local variance is less than 10.

The resulting feature points are ordered by row and column position. A string of digits is constructed by concatenating the column and row positions of these points. This series is encrypted using private key encryption to generate the image signature.
Digital Signature Verification

In order to determine the authenticity of a received image $B$ with respect to the image $A$ that has been sent. The set of feature points $S_B$ is computed from $B$. This is compared with the set of points $S_A$, obtained by decrypting the digital signature. If each feature location present in $S_B$ is also present in $S_A$, the received image is considered as authentic, otherwise reject the image as unauthentic.

Size of Hash

The size of Hash is equivalent to the number of bytes needed to store the locations of the local maxima of feature points detected using the function $P_{i,j}(x) = |M_i(x) - \gamma M_j(x)|$, where $M_i(x)$ and $M_j(x)$ represents the responses of Mexican Hat wavelet at image location $x$ for scales $i$ and $j$ and $\gamma = 2^{-(i-j)}$.

Computation

The total computation is equal to the number of operations in the feature value computation, which also includes the number of operations involved in the computation of variance in a circular neighborhood with a radius of 5 pixels. This again involves $O(n)$ computations.

Size of Key

Since none of the parameters are hidden as keys in the feature value computation, the size of the key is equal to the size of the key of the encryption scheme used.

Probability of Success in Cheating

The probability of success depends on constructing two images, with different impressions, having the same feature codes. The probability of success in finding another image with the same feature code is one (100%).

This can be shown by the following procedure. Find two sets of values in a wavelet transform from scale $i$ to $j$ in a circle of radius 5, which have the variances above a threshold. The centers of those two sets represent the feature points.

If we can replace each feature neighborhood of an image with another, two images which produce the same set of feature codes can be found. This means, the above hash would not be collusion free.
3.10.4 Applying Signatures on Digital Images by I.Pitas and T.H.Kaskalis

The signature scheme proposed by Pitas [42, 43, 44] slightly modifies the intensity level of randomly selected image pixels, and during signature detection mean intensity value of the marked pixel is compared against that of the non-marked pixels using Statistical hypothesis of testing. The authors also suggest that compression robustness can be achieved by selecting random blocks of sizes $2 \times 2$ or $3 \times 3$ instead of selecting random points for marking. More precise version of signature (currently known as watermark) embedding and verification algorithm follows.

**Signature Generation and Verification**

Consider an image of dimensions $N \times M$ represented as follows:

$$I = \{x_{nm}, n \in \{0, ..., N - 1\}, m \in \{0, ..., M - 1\}\}$$

where $x_{nm} \in \{0, 1, ..., L - 1\}$ is the intensity level of pixel $(n, m)$. A signature pattern $S$ is a binary pattern of the same size where the number of ”ones” equals the number of ”zeros”:

$$S = \{s_{nm}, n \in \{0, ..., N - 1\}, m \in \{0, ..., M - 1\}\},$$

$$s_{nm} \in \{0, 1\},$$

It is obvious that, in order to be used for protection, the signature pattern should be known only to the signature owner. Using $S$ we can split $I$ in two subsets of equal size:

$$A = \{x_{nm} \in I, s_{nm} = 1\}$$

$$B = \{x_{nm} \in I, s_{nm} = 0\}$$

$$|A| = |B| = |I|/2 = (N \times M)/2 = P$$

$$I = A \cup B$$

The digital signature is superimposed on the image as follows:

$$C = \{x_{nm} \otimes k, x_{nm} \in A\}$$

where $\otimes$ is a superposition law. The signed image is then given by:
\[ I_s = C \cup B \]

The signature can be considered as a two-dimensional signal \( f_{nm} \):

\[
\begin{align*}
  f_{nm} &= k \text{ if } s_{nm} = 1 \\
  f_{nm} &= 0 \text{ if } s_{nm} = 0
\end{align*}
\]

Let us denote by \( \bar{a}, \bar{b}, \bar{c}, s_a, s_b, \) and \( s_c \) the sample mean values and the sample standard deviations of the pixels belonging to the subsets \( A, B, \) and \( C \) respectively. Signature detection is based on the examination of the difference \( \bar{w} \) of the mean values \( \bar{c}, \bar{b} \):

\[
\bar{w} = \bar{c} - \bar{b}
\]

If the image has been signed then \( \bar{w} \) is close to \( k \) whereas in the case of an unsigned image or an image bearing a signature different from the one that we are looking for, \( \bar{w} \) is approximately zero. In other words, \( \bar{w} \) is a random variable whose mean is zero for an unsigned image and \( k \) for an image that has been signed. The decision on whether the image is signed or not is taken using hypothesis testing. The test statistics \( q \) that has been used is given by

\[
q = \frac{\bar{w}}{\sigma_w}
\]

where:

\[
\sigma_w^2 = \frac{(S_c)^2 + (S_b)^2}{P}
\]

The Null and the Alternative Hypotheses in this case are:

- \( H_0 \): There is no signature in the image.
- \( H_1 \): There is a signature in the image.

Under the null hypothesis the test statistics \( q \) follows a zero mean Student distribution with \((2P - 2)\) degrees of freedom. In order to decide whether the image is signed or not, the value of \( q \) is tested against a threshold \( t \). If \( q > t \) we assume that the image is signed, otherwise we conclude that the image bears no signature. The threshold \( t_{1-\alpha} \) that minimizes both the type I error (accept the existence of a signature although there is none) and the type II error (reject the existence of a signature although there is one) is given by:

\[
t_{1-\alpha} = \frac{k}{2\sigma_w}
\]
The value of $t_{1-\alpha}$ corresponds to a specific certainty level $(1 - \alpha)$ i.e. a probability of correct signature detection. Solving for $k$ we obtain the relationship:

$$k = [2\sigma \tilde{w} t_{1-\alpha}]$$

Therefore, during the signature superposition we can specify the degree of certainty level $(1 - \alpha)$ which we want to have during the signature detection phase and using $k = [2\sigma \tilde{w} t_{1-\alpha}]$, obtain the value of $k$ that will lead to this desired value. In conclusion, the proposed algorithm can be summarized as follows:

**Signature casting:** Generate $S$. Decide for the desired certainty level, calculate $k$ using $k = [2\sigma \tilde{w} t_{1-\alpha}]$ and cast the signature using $C = \{x_{nm} \otimes k, x_{nm} \in A\}$ and $I_s = C \cup B$.

**Signature detection:** Generate $S$. Evaluate $q$ using $\bar{w} = \bar{c} - \bar{b}$, $q = \bar{w} / \sigma_c$, and $\sigma_c^2 = ((S_c)^2 + (S_b)^2) / P$ and compare it with $t_{1-\alpha}$ to decide for the signature existence.

The above method leads to signatures that ensures safe detection with minimum visual distortion.

**Size of MAC**

This is a watermark. Half the pixels are modified. The number of pixels that are marked depends on the size of the image.

**Computation**

The total computation is equal to the operations involved in the computation of mean and standard deviation of two sets of pixels. This again is of order $O(n)$.

**Size of Key**

The size of key is equal to the sum of the size of the seed of Pseudo Random Number Generator for selecting half the pixels for marking and the size of $k$ or the amount of change (mark).

**Probability of Success in Cheating**

The probability of success in cheating is equal to the sum of the probability of choosing exactly the same pixels as that was marked and the probability of guessing the value for $k$ (amount of change). More precisely, the probability for former is given by $1/nC_n/2$. 

3.10.5 A Robust Content Based Digital Signature for Image Authentication by Marc Schneider and Shih-Fu Chang

The method proposed by Schneider [33] uses image block histograms to authenticate the visual content in an image. In the authentication process the Euclidean distance between histograms of each block in the original image and the histogram of the corresponding block in the image under inspection is computed. The sum of all such distances over the entire image is used as a measure of image authenticity. A more precise version of the signature generation and verification algorithms follows.

**Signature Generation**

The steps involved in the signature generation are:

1. Divide an image into blocks of size $m \times n$. These sizes could vary from block to block depending on the importance of the region.

2. Compute the intensity histogram for each block.

3. Group those blocks into pairs. These pairing could be overlapping or non-overlapping depending on the level of security needed.

4. Compute the Euclidean distance for each pair.

5. The ordered set of those Euclidean distances or the sum of those Euclidean distances forms the hash.

The choice of block sizes used, location of those blocks, and the information about block pairing can be used as keys.

**Signature Verification**

The steps involved in the signature verification are:

1. Divide the received image into blocks of size $m \times n$. If these sizes vary from block to block, the choice of block sizes and the locations of those blocks must be known.

2. Compute the intensity histogram for each block.

3. Group those blocks into pairs according to the key information about block pairing.
4. Compute the Euclidean distance for each pair.

5. Compare the ordered set of those Euclidean distances or the sum of those Euclidean distances with the received ones. If they are equal, the received image is authentic. Otherwise reject the image as manipulated.

**Size of Hash**

Suppose n pair of blocks are chosen for Euclidean distance computation, the size of the hash will be equal to n time the number of bytes needed to store a real number.

**Computation**

The computation involved is equal to the sum of operations involved in choosing the block pairs, operations involved in Euclidean distance computation and the operations involved in hash value computation. This again is bounded by the order $O(n)$.

**Size of Key**

If none of the parameters are hidden as keys in Euclidean distance computation, the size of the key will depend on the encryption scheme used. However, the size of the blocks, the block pairs chosen for histograms, and the size of the intervals used in the histograms could have been chosen as keys.

**Probability of Success in Cheating**

The probability of success in cheating is equal to the probability of success in constructing two images with different impressions having the same block histograms.

This can be illustrated as follows. The two images illustrated in Table 3.4 have the same histogram as shown in Table 3.5.

```
<table>
<thead>
<tr>
<th>1</th>
<th>3</th>
<th>5</th>
<th>3</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>
```

| Table 3.4: Two distinct images |

Since none of the parameters are hidden as keys in the hash value computation, the probability of finding another image with the same hash is one (100%).
Table 3.5: Histogram computed

3.10.6 Content-based Digital Signature for Motion Picture Authentication and Content-Fragile Watermarking by Jana Dittmann, Arnd Steinmetz, and Ralf Steinmetz

Dittmann’s method [35] also suggests using Canny’s edge detector for feature extraction. After edge detection, it uses variable length code for data reduction, produces hash value of the variable length code, and finally signs the hash value. During the verification process reverse happens. Although Dittmann says this method survives certain level of compression, it doesn’t exactly indicate the tolerance level. A more precise version of the signature generation and verification algorithm follows.

Signature Generation

The edge-based signature is calculated in the following way: First, the edge characteristics $C_I$ of the image $I$ is extracted using the Canny edge detector $E$ and then it is transformed into a binary edge pattern $EP_{CI}$. Next, a VLC (Variable Length Code) for data reduction is produced as the feature code instead of a hash value and finally the image signature $SigI$ (initialized with a private key of the originator) of the feature code is calculated. The signature generation $sign$ calculates the hash over the VLC code and signs the hash value, instead of using directly the VLC code.

1. Feature extraction: $C_I = E(I)$
2. Binary edge pattern: $EP_{CI} = f(C_I)$.
3. Feature code: $VLC(EP_{CI})$
4. Sign feature code:
   $$SignI = sign(Hash(VLC(EP_{CI})))_{Private Key}$$

The signed VLC code is now added to the image.
Signature Verification

The verification process is performed in the following way: If the user gets the image $T$, he first calculates the actual image edge characteristics $C_T$ and the binary edge pattern $EP_{CT}$ and then he extracts and verifies the signed feature code, the original edge characteristics, with the appropriate public key. Finally, he compares both characteristics.

Verification steps:

1. Feature extraction: $C_T = E(T), EP_{CT} = f(C_T)$

2. Extract original feature code: $EP_{CI} = VLC(EP_{CI}),$ verify $Hash(VLC(EP_{CI})) = Decrypt(SigI)_{PublicKey}$

3. Check $EP_{CI} = EP_{CT}$

Size of Hash

The size of hash is equal to the size of hash of the variable length coding produced on the edge bit map.

The size of edge bit map depends on how the edge bit map is computed. If the whole edge image is used, it will be equal to the size of bytes needed to store the whole image. If only the edge pixel co-ordinates are used, it will be based on the amount of edges in an image.

Computation

The total computation involved is equal to the sum of operations in edge detection, operations in variable length coding, and the operations in hashing. This again involves $O(n)$ operations.

Size of Key

If none of the parameters are hidden as keys in edge detection, variable length coding, or hashing, the size of the key will be equal to the size of the key of the encryption scheme used.

Probability of Success in Cheating

Because the scheme is edge dependent, it will be very hard to construct another image, with exactly the same edge impression, having a different meaning. Colour images can be manipulated by changing the colour without modifying the edges.
Since none of the parameters are hidden as keys in the hash computation, the probability of success in finding another image with the same hash is one (100%).

3.11 Conclusion

In this chapter, we have classified hash functions, defined basic properties of hash functions, described relationship between properties, and explained data integrity and message authentication. Before describing all the previous image authentication schemes with a detail analysis on the size of hash, the computation involved, the size of key space used, and the level of security involved, a section describing image hashing has also been added.
Chapter 4

Statistical Based Image Hashing

4.1 Introduction

The basic idea is to calculate statistical values that capture the main features of an image and remain essentially unchanged through acceptable transformations. Here, *essentially unchanged* means the hash value is considered *close*. For $A$, the set of admissible transformations, we only consider JPEG compression with different quality levels. All other transformations, including filtering, are considered inadmissible manipulations that must be detectable by the system. However the proposed systems are flexible and can be used with other definitions of $A$. The larger transformation set means wider band for acceptance and hence higher rate of false acceptance.

The statistics are collected from specific *regions* of the image. Regions can be obtained through dividing the image into blocks or through a segmentation algorithm such as k-means [48] or by randomly choosing a subset of pixels.

4.1.1 Statistics

We have used the following two statistics.

Mean which is the average of a distribution given by

$$\text{Mean } \mu = \frac{\sum_{i=1}^{n} x_i}{n}$$

Standard deviation, defined as the dispersion of a distribution from its mean and is given by

$$\text{Standard deviation } \sigma = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \mu)^2}{n}}$$

Other possible statistics are *Kurtosis* that is the degree of peakness of a distribution, usually taken relative to a normal distribution, and *Skewness* which is the degree of asymmetry or departure from symmetry of a distribution [49]
4.1.2 Hypothesis Testing

Hypothesis testing for comparing the mean of two normally distributed populations with distinct variances might consist of a null hypothesis \( H_0 : \mu_1 - \mu_2 = 0 \) with an alternative hypothesis \( H_1 : \mu_1 - \mu_2 \neq 0 \). The test statistic which follows the t-distribution is given by \( t = \frac{\overline{x}_1 - \overline{x}_2}{\sqrt{s_1^2/n_1 + s_2^2/n_2}} \) where \( \overline{x}_1 \) and \( \overline{x}_2 \) are the sample means, \( s_1 \) and \( s_2 \) are the sample variances, and \( n_1 \) and \( n_2 \) are the sample sizes of the respective populations.

4.1.3 K-mean/LBG Algorithm

K-means was developed for clustering procedure in pattern recognition and it can be described as follows.

Given a large set of output vectors from the source known as the training set, and an initial set of \( k \) representative patterns, assign each element of the training set to the closest representative patterns. After all elements are assigned, the representative pattern is updated by computing the centroids of the training set vectors assigned to it. When the assignment process is complete, we will have \( k \) groups of vectors clustered around each of the output points.

Based on k-mean algorithm, Linde-Buzo-Gray (LBG) [48] developed the following algorithm which is well known as LBG Algorithm.

1. Start with an initial set of reconstruction values \( \{Y_i^{(0)}\}_{i=1}^M \) and a set of training vectors \( \{X_n\}_{n=1}^N \). Set \( k = 0 \), \( D^{(0)} = 0 \), select threshold \( \epsilon \).

2. The quantization regions \( \{V_i^{(k)}\}_{i=1}^M \) are given by

\[
V_i^{(k)} = \{X_n : d(X_n, Y_i) < d(X_n, Y_j) \forall j \neq i, i = 1, ..., M\}
\]

Assume that none of the quantization regions are empty.

3. Compute the average distortion \( D^{(k)} \) between the training vectors and the representative reconstruction values.

4. If \( \frac{D^{(k)} - D^{(k-1)}}{D^{(k-1)}} \leq \epsilon \) stop, otherwise continue.

5. \( k = k + 1 \). Find the new reconstruction values \( \{Y_i^{(k)}\}_{i=1}^M \) that are the average value of the elements of the quantization regions \( V_i^{(k-1)} \). Go to step 2.
4.2 Image Hash

4.2.1 Hash Generation and Verification

Our hash generation and verification falls into following categories. First two methods are based on region statistics: mean, standard deviation, kurtosis and skewness, and the third method is based on a codebook trained on compressed images feature vectors using LBG algorithm. The method 4 is based on statistical hypothesis of testing and the method 5 is based on edge detection.

4.2.2 Method-1: Block Statistics

Hash Generation

Let $A_{\sigma}$ consists of all JPEG compression algorithm down to quality level $\sigma$.

1. Divide the original image $I_o$ into square blocks of size $a \times b$. Assume there are $m \times n$ blocks.

2. For each block, compute one of the above mentioned statistics $S_{ij}^o$, $i = 1,...,m$, $j = 1,...,n$.

3. For all images $I_m = a(I), a \in A_{\sigma}$, perform steps 1 and 2 above to obtain $S_{ij}^m$.

4. Find the difference in statistics of the modified image block from that of the corresponding original image block $|S_{ij}^o - S_{ij}^m|$

5. For all $i = 1,...,m$, $j = 1,...,n$, find $\tau = \max_{a \in A_{\sigma}} |S_{ij}^o - S_{ij}^m|$ and form the threshold $T_{\sigma} = \tau$.

The hash value for an image consist of the sequence of statistics of the original image blocks $S_{ij}^o$, block size $a \times b$, type of statistics used and the threshold $T_{\sigma}$ determined above. The block size determines the level of coarseness of the statistics. Choosing small $a \times b$ will produce long signatures gathering the detail of the image while choosing large $a \times b$ will produce short signatures losing the detail of the image.

Hash Verification

To determine if a candidate image $I_c$ has the same hash $x_I$ as the image $I$ follow the steps below.

1. Perform steps 1 and 2 on $I_c$. 

2. Calculate $|S^o_{ij} - S^c_{ij}|, i = 1, ..., m, j = 1, ..., n$.

3. If $|S^o_{ij} - S^c_{ij}| \leq T_\sigma, i = 1, ..., m, j = 1, ..., n$ then the $I_c$ is said to collide with $I$.

These statistical computations can also be done in the transform domain after applying DCT or Wavelet transforms at different sub-bands.

**Experiments to Verify Method-1**

Experiments were performed on ten different images: Lena, Water fall, Peppers, Paper machine, Opera, Pills, Beach, Wild flower, Corrosion, Fontain. Original image was compressed at different quality levels 70, 50, 25, and 10, filtered using Gaussian, Median and FMLR (Frequency Mode Laplacian Removal) filters, and rotated to -0.5 degrees. Some intentional manipulations such as removal of a portion of an image and drawing a line across were also performed. The software provided by Stirmark was used to do all these manipulations. The experiments were performed using the mean difference of the image blocks and the difference of standard deviation of the image blocks. The images used were recommended by Stirmark and was found in [47].

The experimental results showed that on all images except for Corrosion and Wild flower, maximum difference in statistics at compression level 25 or 10 could be used as the threshold to distinguish between compression, and other types of manipulations such as Median filtering, Gaussian filtering, FMLR attack, and rotation. FMLR attack produced small maximum differences in statistics on certain images due to which some amount of compression was sacrificed. In most cases standard deviation performed better than mean in the sense that choosing standard deviation compensated more compression than choosing mean. Thus we recommend to use standard deviation instead of mean. We also noticed that except for intentional manipulations rotation produced the biggest maximum difference is statistics.

**4.2.3 Possible Attacks**

Our hash algorithm produces mean or standard deviation of image blocks of size $n \times n$ as hash values. By incorporating both mean and standard deviation, the security of the scheme can be enhanced. The scheme is extremely vulnerable if only one of the statistics is used. Including more than one statistics will further enhance the security of the hash function and so we recommend to use Kurtosis and Skewness in addition to mean and standard deviation.
Attacks

If either mean or standard deviation is used in the hash value, it is always possible to reconstruct another set of pixel values that produces the same mean or standard deviation. The following examples illustrate these attacks:

- Suppose $x_1, x_2, \ldots, x_n, x_{n+1}, \ldots, x_{2n}$ are the pixel values involved and $\bar{x}$ is the mean of these values, it is always possible to reconstruct another set of pixel values by adding 1 to half the pixel values and subtracting 1 from rest of the pixels to produce the same mean, for example, $x_1 + 1, x_2 + 1, \ldots, x_n + 1, x_{n+1} - 1, \ldots, x_{2n} - 1$. The value 1 could be replaced with any constant $c$ to produce the same mean.

- Suppose $x_1, x_2, \ldots, x_n$ are the pixel values with mean $\bar{x}$. Adding 1 to all the pixel values will produce a new set of pixel values $x_1 + 1, x_2 + 1, \ldots, x_n + 1$ with mean $\bar{x} + 1$. Computing the standard deviation for both $x_1, x_2, \ldots, x_n$ and $x_1 + 1, x_2 + 1, \ldots, x_n + 1$ will produce the same value. Here also the value 1 could be replaced with any constant $c$ to produce the same standard deviation. Note also that increasing or decreasing all pixel values by 1 wouldn’t change the image impression very much.

- Let us now see what happens when both mean and standard deviation are used as hash value.

(a) Suppose there are only 2 pixels $x_1$ and $x_2$ in a block, such that $x_1 + x_2 = 2\bar{x}$ then $(x_1 - \bar{x}) = (\bar{x} - x_2)$—(1)
and $(x_1 - \bar{x})^2 + (x_2 - \bar{x})^2 = 2\sigma^2$—(2)
Substituting (1) in (2) will result in $2(x_1 - \bar{x})^2 = 2\sigma^2$, which in turn will give $x_1 = \bar{x} - \sigma$ and $x_2 = \bar{x} + \sigma$.
In the case of two pixel values, for a fixed mean, the standard deviation values of the pixels are uniquely determined as shown above.

(b) Suppose there are three pixels $x_1, x_2$, and $x_3$ in a block,$x_1 + x_2 + x_3 = 3\bar{x}$—(1)
and $(x_1 - \bar{x})^2 + (x_2 - \bar{x})^2 + (x_3 - \bar{x})^2 = 3\sigma^2$. —(2)
(1) results in $(x_1 - \bar{x}) = [(\bar{x} - x_2) + (\bar{x} - x_3)]$—(3)
Substituting (3) in (2) will give
$2(x_2 - \bar{x})^2 + 2(x_3 - \bar{x})^2 + 2(x_2 - \bar{x})(x_3 - \bar{x}) = 3\sigma^2$—(4)
If the quadratic equation (4) is solvable for a fixed $x_2$ there will be at most 2 solutions for $x_3$ ($x_1$). Since $x_2$ can take 256 values, there can be at most 512 solutions
for $x_3$ ($x_1$). That is, with 3 pixels there can be atmost 512 combination of values that could produce the same mean and standard deviation.

**Security of the Scheme**

We have already analysed the security of the scheme with 3 pixel values. Even with 3 pixel values there are several combinations that produce the same mean and standard deviation. Out of these several possibilities there can be some combinations which may be very close to the original set of values. If such combinations are chosen image impression would’t change much from the original.

If the number of pixel values increase, the possibility of constructing another image with the same impression which hashes to the same mean and/or standard deviation will also increase. By looking at all these possible attacks, it becomes evident that we should make this scheme key dependent.

**Key Dependent Scheme**

There are several possibilities of making this scheme key dependent.

- Choose random sub set of blocks of size $a \times b$ in the hash computation. — Positions of the blocks become the key.

- Hide the size of blocks. — Size of block becomes the key.

- Choose random sub-regions corresponding to predetermined shapes and sizes in hash computation— Positions and sizes of the shapes become the key.

- Choose different statistical measures for each shape (Circles, Squares and triangles) — Positions, size of the shapes and the type of statistics used become the key

**4.2.4 Method -2: Region Statistics**

**Hash Generation**

Our hash generation and verification is based on the statistical measures and the k-mean clustering discussed before.

1. The original image $I_o$ is segmented into $P$ regions using the k-mean algorithm and a code book trained on the original and compressed images’ feature vectors comprising of pixel value, mean of adjacent pixels, and standard deviation of adjacent pixels.
2. Compute the statistics $S_i^0$ for each region $i = 1, \ldots, P$.

3. For all images $I_m = a(I), a \in A$ use the codebook above to segment the image and find the statistics $S_i^m$.

4. Find the Euclidean distance $D_m$ between the statistics of the modified image regions $S_i^m$ and the statistics of the original image regions $S_i^0$, $D_m = \sqrt{\sum (S_i^0)^2 - (S_i^m)^2}$.

5. Determine a threshold $T$ based on the value of $D_m$ for a number of compressed images $I_m = a(I), a \in A$.

6. The most appropriate value for $T$ would be the maximum of all $D_m$ computed in the above step.

The hash value consists of the sequence of statistics of the original image regions $S_i^0$, number of regions $P$, type of statistics used, code book and the thresholds $T$ determined above.

**Hash Verification**

To see if the hash value of a candidate image $I^c$ is the same as the hash value of the original image follow the steps below.

1. Segment the image $I^c$ into $P$ regions using the k-mean algorithm and the received code book.

2. Compute the required statistics $S_i^c$ of those regions.

3. Find the Euclidean distance $D_c$ between the statistics of $I^c$ and $I$, and if $D_c \leq T$, decide the two hash values are the same, otherwise the hash values are different.

The intuition behind Method-2 is that the codebook is obtained by taking all acceptable modifications of the image into account. Now if the candidate image is from an acceptable modification, the segments are very close to the segments of the original image and the statistics of the regions are close to the statistics of the regions of the original image. Hence the Euclidean distance between the statistics of the corresponding segments having the same centroids of two images are very small. If the image is from an inadmissible transformation, the regions are different and the distance is high.
Experiment to Verify Method-2

The experiments were performed after training the codebook on feature vectors comprising of pixel value, mean of adjacent pixels, and standard deviation of adjacent pixels using the original and all compressed images’ pixels. The reason for choosing only compressed images’ feature vectors in training the codebook was to make sure the centroids were close to compressed images’ feature vectors than to any other modified images’ feature vectors. The LBG algorithm was used in the training process to produce the best centroids or codebook and then the k-mean algorithm was used in segmenting the images.

Experiments were performed on ten different images: Lena, Water fall, Peppers, Paper machine, Opera, Pills, Beach, Wild flower, Corrosion, Fontain. The original image was compressed at different quality levels 70, 50, 25, and 10, filtered using Gaussian, Median and FMLR (Frequency Mode Laplacian Removal) [46] filters, and rotation to -0.5 degrees. Initial centroids were trained on only compressed images’ feature vectors using LBG algorithm. By doing this we assumed the centroids were minimized in mean square distance to compressed images’ feature vectors than to any other modified images’ feature vectors. Images were segmented into 15 and 30 regions and the Euclidean distances between the original image’s region statistics and the modified images’ region statistics were also computed.

Using compression at quality level of 25 as the threshold, most of the modifications were detected except for Median filtering on all images other than Opera and Wildflower, and FMLR attack on Pappers, Lena and Waterfall images. To validate method-2, we computed the total sum of square distances (TSSD) between the centroids (trained on all compressed images’ feature vectors) and their nearest feature vectors or class vectors for each modification. What we observed was that TSSD computed on all compressed and original images were close to each other. In some cases TSSD for median filtering fell within the range of TSSD computed on compressed images. This explains why Median filtering was not detected on some images.

We guess if the regions created on all images (original, compressed and modified) using a codebook minimised the number of pixels falling from one visual region to the other, our scheme would have performed well distinguishing compression from other modifications. Choosing such a codebook after many experiments would make sure our scheme is a secure authentication scheme.
A Possible Attack

For this attack, let us assume that the image has been divided into 3 regions with centroids \( c_1, c_2, \) and \( c_3 \), each region has only two pixels, and mean of pixel values in each region is used as the statistics.

- Let \( c_1, c_2, \) and \( c_3 \) be the centroids trained on original and compressed images feature vectors.

- Let \( \overline{x}_1, \overline{x}_2, \) and \( \overline{x}_3 \) be the means of region pixels of the original image \( X \) having centroids \( c_1, c_2, \) and \( c_3 \), respectively.

- Let \( \overline{y}_1, \overline{y}_2, \) and \( \overline{y}_3 \) be the means of region pixels of the compressed image \( Y \) (compressed to acceptable quality level) having centroids \( c_1, c_2, \) and \( c_3 \), respectively.

- Threshold \( T = \sqrt{\frac{\sum (\overline{x}_i - \overline{y}_i)^2}{3}} \)

- The hash value comprises of \( c_1, c_2, c_3, \overline{x}_1, \overline{x}_2, \overline{x}_3 \) and \( T \).

**Attack** Let us construct an attack image \( Z \) with region means \( \overline{x}_1, \overline{x}_2, \) and \( \overline{x}_3 \) such that it has a different impression to \( X \) and the Euclidean distance computed between the region statistics of \( X \) and \( Z \) is less than the threshold \( T \). That is, \( ED = \sqrt{\sum (\overline{x}_i - \overline{y}_i)^2/3} < T \).

Suppose \( \overline{x}_1 = (x_{11} + x_{12})/2, \overline{x}_2 = (x_{21} + x_{22})/2, \) and \( \overline{x}_3 = (x_{31} + x_{32})/2. \) Let us choose pixel values for the three regions \( z_{11}, z_{12} \) (region 1), \( z_{21}, z_{22} \) (region 2), and \( z_{31}, z_{32} \) (region 3) which differ from original pixel values of image \( X \) such that the following holds:

\[
\begin{align*}
\overline{z}_1 &= (z_{11} + z_{12})/2 = \overline{x}_1 \\
\overline{z}_2 &= (z_{21} + z_{22})/2 = \overline{x}_2 \\
\overline{z}_3 &= (z_{31} + z_{32})/2 = \overline{x}_3 
\end{align*}
\]

Since the region means of attack image \( Z \) are the same as the region means of original image \( X \), \( ED \) will be equal to 0 and less than the threshold \( T \). Thus, \( Z \) will be considered as authentic.

Even if the statistics used are standard deviations of the regions, we can construct attack images which are proved to be authentic. Therefore, these hash values must be signed or encrypted by proper cryptographic algorithm to provide better security.
4.2.5 A Comparison

- Method-1 uses uniform blocks among each modification while the regions arising from k-mean segmentation of method-2 are not uniform among each modification.

- Method-1 performs better because it compares the statistics of corresponding blocks of the same size while the method-2 is considered to be inferior to method-1 since the regions segmented by k-mean may not remain the same for each modification. Carefully choosing the centroids which produce the same visual regions may enhance performance.

- Method-1 is easy to implement and the system is relatively fast where as method-2 is too costly in terms of time and computation due to k-mean segmentation.

- Increasing the number of blocks or decreasing the block size improves the performance of method-1 at the expense of long hash where as increasing the number of regions does not impact much on the performance of method-2.

4.2.6 Method-3: K-means

In this section, we describe a hashing scheme which uses the codebook trained on feature vectors of images compressed above an acceptable quality level.

Hash Generation

1. Take an image $I$ and JPEG compress it at different quality levels $I_1, I_2, ..., I_k$.

2. Find the feature vectors comprising of pixel value ($p$), mean of adjacent pixels ($\bar{x}$), and standard deviation of adjacent pixels ($\sigma$) for all of the above images. This forms the training set $T_c$ for LBG. This implies if $(p, \bar{x}, \sigma) \in T_\alpha$ then $(p, \bar{x}, \sigma) \in T_c = T_{c_1} \cup T_{c_2} \cup ... \cup T_{c_k}$ where $T_{c_i}$ is the training set comprising of feature vectors of compressed image $I_i$.

3. Train the initial codebook $IC$ on training set $T_c$ created above using only compressed images’ feature vectors. By doing this we assume the centroids are minimised in mean square distance to compressed images feature vectors than to any other modified images feature vectors.

4. Let $FC = \{C_1, C_2, ..., C_p\}$ be the final codebook created in the last step.
5. Starting with the codebook \( FC \) created above make one more iteration on LBG Algorithm firstly using the training set \( T_{ci} \) created with the feature vectors of a compressed image \( I_i \). Suppose \( FC' = \{ C'_1, C'_2, \ldots, C'_p \} \) is the codebook after one iteration, note the deviation caused in Euclidean distance \( ED_{comp} = \sqrt{\sum_{i=1}^{p}(C_i - C'_i)^2} \).

6. Repeat step 5 again using the training set \( T_f \) created with the feature vectors of a modified or filtered image \( I_f \). Suppose \( FC'' = \{ C''_1, C''_2, \ldots, C''_p \} \) is the codebook after one iteration, note the deviation caused in Euclidean distance \( ED_{other} = \sqrt{\sum_{i=1}^{p}(C_i - C''_i)^2} \).

7. By looking at the minimum and maximum values of \( ED_{comp} \) for all compressed images \( I_i, i = 1, \ldots, p \), determine a range for compression tolerable Euclidean distances \([a, b]\).

8. Our hash of an image comprises of \( FC', a, \) and \( b \). Size of the hash is given by \((p\times3+2)\times r\) bytes, where \( p \) is the size of codebook and \( r \) is the number of bytes used to represent a real number.

**Hash Verification**

To check if a candidate image \( I' \) has the same hash value as \( I \).

1. Produce a sequence of feature vectors comprising of each pixel value, mean of its adjacent pixels, and standard deviation of its adjacent pixels.

2. Run one more iteration of LBG on the received codebook \( FC \). Suppose \( FC'' = \{ C'''_1, C'''_2, \ldots, C'''_p \} \) is the codebook after one iteration, note the deviation caused in Euclidean distance \( ED_{received} = \sqrt{\sum_{i=1}^{p}(C_i - C'''_i)^2} \).

3. If the computed Euclidean distance \( ED_{received} \) falls in the range \([a, b]\), the candidate image has the same hash value, else the image has a different value.

**Experiment to Verify Method-3**

To verify the above scheme, we performed experiments on 5 different images: Lena, Fontain, Opera, Paper, and Peppers using 5 different code books. Following are the observations noted. On all images except for Fontain image, compression was clearly distinguished from other modifications. The effect on increase in the size of the codebook (number of regions) was noted. With an increase in the size of the codebook, Euclidean distances also increased, though there were some exceptions. On Peppers
image, Median filtering wasn’t distinguished from compression even on different sized codebooks. Sharpening showed the highest deviation on all images. On some images, as the size of the codebook increased, original image showed only a slight deviation in Euclidean distance from compressed images. Over all the performance was great. Although the size of the codebook did not impact much on the scheme we recommend to choose codebooks of size greater than 10 to make the scheme complex and less vulnerable. Deviation caused to the codebook or centroids is the deviation caused on the mean of region vectors, thus making the scheme statistically dependent. The size of hash is given in the table below assuming 4 bytes are used to represent a real number. The effect on choosing different codebooks was also observed. The results indicated choosing different codebooks had no impact on the scheme.

<table>
<thead>
<tr>
<th>Size of codebook</th>
<th>Size of hash</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>$(3*15+2)^4=178$ bytes</td>
</tr>
<tr>
<td>30</td>
<td>$(3*30+2)^4=368$ bytes</td>
</tr>
<tr>
<td>45</td>
<td>$(3*45+2)^4=548$ bytes</td>
</tr>
<tr>
<td>60</td>
<td>$(3*60+2)^4=728$ bytes</td>
</tr>
</tbody>
</table>

Table 4.1: Size of hash

### 4.2.7 Method-4: Hypothesis of Testing

The scheme is based on the assumption that the hash algorithm and the verifier of the image have the same random number generators. That is, the hash function and the verification algorithm are able to produce the same random co-ordinates of the image.

#### Hash Generation

1. Random number generator generates a sequence of $n_1, n_2, ..., n_p$ co-ordinates ($p$-sets).

2. Calculate the mean and variance of pixel values for each set of image co-ordinates of the original image. $(\mu_1, \sigma_1^2, \mu_2, \sigma_2^2, ..., \mu_p, \sigma_p^2)$

3. Calculate the mean and variance of pixel values for each set of image co-ordinates of the JPEG compressed image at quality level 25 (Acceptable compression - determined by prior experiments). $(\mu_1^2, \mu_2^2, ..., \mu_p^2)$

4. Calculate $T_i = \frac{\sqrt{\mu_i - \bar{\mu}}}{\sqrt{\frac{\sigma_i^2}{n_i} + \frac{n_i \mu_i^2}}}$ for $i = 1, ..., p$
5. Signature=Encrypt(μ_1, σ^2_1, n_1, T_1, ..., μ_p, σ^2_p, n_p, T_p)

Hash Verification

1. Decrypt the signature to find μ_1, σ^2_1, n_1, T_1, ..., μ_p, σ^2_p, n_p, T_p

2. Random number generator generates a sequence of n_1, n_2, ..., n_p co-ordinates (same as the signer).

3. Calculate the mean and variance of pixel values for each set of co-ordinates of the image to be authenticated. (μ_1β^2_1, μ_2β^2_2, ..., μ_pβ^2_p)

4. Calculate t_i = \frac{|\overline{x}_i - \overline{z}_i|}{\sqrt{\frac{\overline{x}_i^2}{n_i} + \frac{\overline{z}_i^2}{n_i}}} for i = 1, ..., p

5. Suppose all t_i are zero then the original image has not been tempered at all.
   If 0 < t_i ≤ T_i for all i then the original image has been compressed to acceptable level.
   If t_i > T_i for some i then the image has been modified.

Discussion

Reasonable values for n_1, n_2, ..., n_p are to be chosen. If we are going to approximate with normal distributions, n_i have to be larger than 30. We choose p number of random samples instead of one sample. Value for p should be based on the length of the signature. Although this scheme somewhat resembles I.Pitas and T.M.KasKalis [42, 43, 44] method of signing an image, it differs from their scheme in the following respect.

- Their scheme embeds the signature in the image itself.
- They do not give any tolerance level for compression.
- Their scheme was based on just one sample where as ours take multiple samples into consideration before making a decision.

Experiments to Verify Method-4

We calculated the test statistics t_i for compressed images at quality levels 75, 50, 25 and 0, FMLR, Gaussian and Median filtered images, rotated image and intentionally manipulated images. We used Lena image for this experiment. Half the image pixels were randomly chosen 10 times to produce these statistics and plotted as shown below.
Figure 4.1: Comparison of thresholds

One can easily note that the images compressed up to the quality level of 25 lie close to each other than the other manipulated images, which clearly indicates the thresholds obtained at this level of compression will allow us to distinguish between acceptable level of compression and any other manipulations.

4.2.8 Method-5: Edge Detection

One of the edge detection scheme that was proposed is based on the variance of the moving 2x2 window over an image. This scheme can be modified to produce a scaled edge image.

MAC Generation

1. Divide an image into 2x2 pixel blocks,

2. Compute the variance of each block,

3. Substitute each 2x2 pixel block with a black and white pixel value depending on whether the variance is above a certain threshold or not.
This scaled edge image can again and again be compressed/scaled until the desired size edge image is produced. MAC of the image is obtained by

1. substituting the black pixel of the scaled edge image with a binary digit 1 and the white pixel of the scaled edge image with a binary digit 0 and

2. making the block size nxn equal to 2x2, 2x3, 3x2, or 3x3, threshold T at each iteration, and the number of iterations as the secret keys

**MAC Verification**

Once the image and its MAC have been received, knowing the key, MAC of the received image can be computed and compared against the received MAC. Verification is based on string matching process.

If the number of matches is above certain percentage, the received image is considered to be authentic, else reject the image as tempered.

**Experiments to Verify Method 5**

The experiments were carried out on five different images by fixing the block size to 2x2, and the threshold T to 250 (SS) at each iteration, and the number of iterations to 4. Number of matches were compared. We experimented the scheme on Lena, Waterfall, Opera, Papermachine, and Peppers images. Using compression at quality level of 50, all modifications (compression at quality level of 25 and 10, FMLR attack, Gaussian filtering, Median filtering, Rotation, Embossing, and Painting) except Median filtering had been detected on all images. FMLR was not detected on Paper machine image.

Since edge pixes are approximated at each iteration, the MAC produced by this scheme may not detect small manipulations of the edges. A possible improvement is to scale down the image by interpolation and apply our edge detection once. This may reduce the approximations we made at each iteration.

We performed this experiment on Corrosion, Fontain, Lena, and Mountain images by scaling the image into one forth of the original size and then applying the edge detection once on the scaled image. Except for Median filtering on Fontain and Mountain images and Gaussian filtering on Mountain image all the modifications were detected.

**4.2.9 Performance Analysis**

Using the above methods to decide on the hash value of an image may result in an incorrect result. To determine a confidence level on the decision we use the following
Table 4.2 shows the maximum block statistics difference $T$ for compressed Waterfall images and the maximum block statistics difference for other manipulated Waterfall images. We could also compute the probability of accepting $H_0$ when actually an image was manipulated, $\gamma$. This indicates the rejection region for $H_0$, which would result in the range of $[a, b]$.

We could also compute the probability of $H_0$ when the image was manipulated, $\beta$. Figure 4.2 shows the range and $\beta$ computed in the following sections are approximates because we have not considered enough sample data.

1. Compute $\gamma$ on Euclidean distances (method 1) for compressed images.

2. Compute $\gamma$ and $\beta$ on Euclidean distances (methods 2 and 3) or maximum difference in block statistics (method 1) for compressed images.

$H_0$ when it should be accepted (i.e., $Pr(\gamma = \alpha)$) is the level of significance of the test of $\gamma$. The probability of accepting $H_0$ when actually an image is manipulated is given by area $\beta$.

Assuming the level of significance is $\alpha = 0.01$ and using the following statistics:

- Hypothesis $H_1$: Image under consideration is rejected as manipulated if the Euclidean distance (methods 2 and 3) or maximum difference in block statistics (method 1) computed falls out of the range of $[a, b]$.

- Hypothesis $H_0$: Image under consideration is accepted as original or compressed to permissible quality level if the Euclidean distance (methods 2 and 3) or maximum difference in block statistics (method 1) computed falls within the range of $[a, b]$.

Steps: Assume Euclidean distances of hash values (methods 2 and 3) or maximum difference in block statistics of images belonging to admissible set (method 1), follow normal distributions with mean $\gamma$ and standard deviation $\sigma$, i.e., for compression, and the Euclidean distances of hash values (methods 2 and 3) or maximum difference in block statistics (method 1) of images belonging to admissible set, follow normal distribution with mean $\gamma$ and standard deviation $\sigma$, i.e., for other manipulations. We shall now define a null hypothesis and an alternative hypothesis as follows.
Figure 4.2: $\alpha$, $\beta$, and the acceptance region $[a, b]$ assuming the values follow normal distributions with means $\mu$ and $\bar{\mu}$ and standard deviations $\sigma_c$ and $\sigma_o$ respectively.

images. These values were computed after dividing the images into 1200, 300, and 12 equal size blocks and then finding the maximum difference in statistics.

Assuming the level of significance $\alpha = 0.01$ and using the data given in Table 4.2, the acceptance region for $H_0$ and the probability of accepting $H_0$ when actually an image was manipulated, $\beta$, had also been calculated and the results are given in Table 4.2.

Method-2

Table 4.3 shows the values of $D_m$ for compressed Peppers images and the values of $D_m$ for other manipulated Peppers images. These values were computed for images segmented into 15 and 30 regions.

Assuming the level of significance $\alpha = 0.01$ and using the data given in Table 4.3, the acceptance region for $H_0$ and the probability of accepting $H_0$ when actually an image was manipulated, $\beta$, had also been calculated and the results are given in Table 4.3.

Method-3

Table 4.4 shows the $ED_{comp}$ for compressed Lena images and the $ED_{other}$ for other manipulated Lena images. These values were computed using five different codebooks trained only on compressed images feature vectors.

Assuming the level of significance $\alpha = 0.01$ and using the data given in Table 4.4, the acceptance region for $H_0$ and the probability of accepting $H_0$ when actually an
<table>
<thead>
<tr>
<th>Image</th>
<th>Mean</th>
<th>Mean</th>
<th>Mean</th>
<th>Std. Dvi</th>
<th>Std. Dvi</th>
<th>Std. Dvi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blocks</td>
<td>1200</td>
<td>300</td>
<td>12</td>
<td>1200</td>
<td>300</td>
<td>12</td>
</tr>
<tr>
<td>Comp70</td>
<td>4.28</td>
<td>1.44</td>
<td>0.19</td>
<td>30.80</td>
<td>23.96</td>
<td>7.79</td>
</tr>
<tr>
<td>Comp50</td>
<td>5.59</td>
<td>2.14</td>
<td>0.38</td>
<td>32.79</td>
<td>23.97</td>
<td>19.84</td>
</tr>
<tr>
<td>Comp25</td>
<td>7.2</td>
<td>3.13</td>
<td>0.53</td>
<td>44.35</td>
<td>34.41</td>
<td>42.12</td>
</tr>
<tr>
<td>Comp10</td>
<td>11.48</td>
<td>6.21</td>
<td>1.03</td>
<td>78.05</td>
<td>52.62</td>
<td>73.62</td>
</tr>
<tr>
<td>FMLR</td>
<td>7.48</td>
<td>4.36</td>
<td>0.73</td>
<td>53.13</td>
<td>77.22</td>
<td>264.08</td>
</tr>
<tr>
<td>Gauss</td>
<td>16.44</td>
<td>7.05</td>
<td>0.99</td>
<td>74.75</td>
<td>99.96</td>
<td>220.61</td>
</tr>
<tr>
<td>Median</td>
<td>19.64</td>
<td>11.99</td>
<td>4.43</td>
<td>86.23</td>
<td>72.29</td>
<td>86.21</td>
</tr>
<tr>
<td>Rotate</td>
<td>83.8</td>
<td>63.32</td>
<td>8.51</td>
<td>257.94</td>
<td>294.81</td>
<td>77.73</td>
</tr>
<tr>
<td>$\bar{\tau}$</td>
<td>5.69</td>
<td>2.23</td>
<td>0.36</td>
<td>35.98</td>
<td>27.45</td>
<td>23.25</td>
</tr>
<tr>
<td>$\sigma_{\tau}$</td>
<td>1.46</td>
<td>0.85</td>
<td>0.17</td>
<td>7.32</td>
<td>6.03</td>
<td>17.42</td>
</tr>
<tr>
<td>$\bar{\sigma}$</td>
<td>19.83</td>
<td>13.27</td>
<td>2.24</td>
<td>78.58</td>
<td>85.27</td>
<td>103.178</td>
</tr>
<tr>
<td>$\sigma_{\sigma}$</td>
<td>29.19</td>
<td>22.46</td>
<td>3.14</td>
<td>86.83</td>
<td>99.97</td>
<td>102.29</td>
</tr>
<tr>
<td>Rrange[a,b]</td>
<td>[0-9.14]</td>
<td>[0-4.24]</td>
<td>[0-0.77]</td>
<td>[0-53.25]</td>
<td>[0-41.68]</td>
<td>[0-64.35]</td>
</tr>
<tr>
<td>$\beta$</td>
<td>0.36</td>
<td>0.34</td>
<td>0.32</td>
<td>0.39</td>
<td>0.34</td>
<td>0.35</td>
</tr>
</tbody>
</table>

Table 4.2: $T_{\tau}$ using maximum of mean and standard deviation on 1200, 300, and 12 square blocks

image was manipulated, $\beta$, had also been calculated and the results are given in Table 4.4.

### 4.3 Conclusion

We have proposed five schemes for authenticating images surviving acceptable level of content modifications such as JPEG compression. These schemes were based on basic statistical measures, and k-mean segmentation of images. The performance of the first three methods were statistically analysed. Our experiments show that the results are promising on some images. Further experiments and testing are needed to improve the schemes.
<table>
<thead>
<tr>
<th>Image</th>
<th>Mean</th>
<th>Std. Dvi.</th>
<th>Mean</th>
<th>Std. Dvi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regions</td>
<td>15</td>
<td>15</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>Comp70</td>
<td>1.517</td>
<td>16.938</td>
<td>6.358</td>
<td>67.151</td>
</tr>
<tr>
<td>Comp50</td>
<td>2.486</td>
<td>47.080</td>
<td>12.408</td>
<td>85.277</td>
</tr>
<tr>
<td>Comp25</td>
<td>4.188</td>
<td>94.835</td>
<td>7.949</td>
<td>48.421</td>
</tr>
<tr>
<td>Comp10</td>
<td>5.224</td>
<td>99.084</td>
<td>9.419</td>
<td>79.207</td>
</tr>
<tr>
<td>FMLR</td>
<td>7.888</td>
<td>92.36</td>
<td>9.596</td>
<td>102.573</td>
</tr>
<tr>
<td>Gauss</td>
<td>8.314</td>
<td>180.385</td>
<td>16.619</td>
<td>144.84</td>
</tr>
<tr>
<td>Median</td>
<td>2.3821</td>
<td>41.898</td>
<td>17.449</td>
<td>86.959</td>
</tr>
<tr>
<td>Rotate</td>
<td>19.708</td>
<td>1024.92</td>
<td>66.795</td>
<td>5759.25</td>
</tr>
<tr>
<td>Sharp</td>
<td>31.085</td>
<td>1112.94</td>
<td>44.139</td>
<td>1201.15</td>
</tr>
<tr>
<td>$\overline{\sigma}$</td>
<td>2.73</td>
<td>52.95</td>
<td>8.91</td>
<td>66.94</td>
</tr>
<tr>
<td>$\sigma_c$</td>
<td>1.35</td>
<td>39.278</td>
<td>3.136</td>
<td>18.43</td>
</tr>
<tr>
<td>$\overline{\sigma}$</td>
<td>10.657</td>
<td>364.51</td>
<td>23.43</td>
<td>1053.43</td>
</tr>
<tr>
<td>$\sigma_o$</td>
<td>10.98</td>
<td>485.05</td>
<td>23.55</td>
<td>2116.96</td>
</tr>
<tr>
<td>Range[a,b]</td>
<td>[0-5.92]</td>
<td>[0-145.65]</td>
<td>[0-16.31]</td>
<td>[0-110.44]</td>
</tr>
<tr>
<td>$\beta$</td>
<td>0.33</td>
<td>0.32</td>
<td>0.38</td>
<td>0.33</td>
</tr>
</tbody>
</table>

Table 4.3: $D_m$ using mean and standard deviation on 15 and 30 regions

<table>
<thead>
<tr>
<th>Image</th>
<th>Set-1</th>
<th>Set-2</th>
<th>Set-3</th>
<th>Set-4</th>
<th>Set-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comp90</td>
<td>210.44</td>
<td>228.61</td>
<td>224.09</td>
<td>634.55</td>
<td>129.11</td>
</tr>
<tr>
<td>Comp70</td>
<td>143.82</td>
<td>188.92</td>
<td>190.91</td>
<td>613.25</td>
<td>116.97</td>
</tr>
<tr>
<td>Comp50</td>
<td>120.72</td>
<td>214.47</td>
<td>205.86</td>
<td>597.92</td>
<td>99.80</td>
</tr>
<tr>
<td>Comp30</td>
<td>123.12</td>
<td>193.84</td>
<td>171.88</td>
<td>592.89</td>
<td>87.23</td>
</tr>
<tr>
<td>Comp10</td>
<td>86.67</td>
<td>177.37</td>
<td>189.57</td>
<td>251.02</td>
<td>117.32</td>
</tr>
<tr>
<td>FMLR</td>
<td>2618.15</td>
<td>342.55</td>
<td>327.28</td>
<td>712.94</td>
<td>288.16</td>
</tr>
<tr>
<td>Gauss</td>
<td>563.57</td>
<td>500.29</td>
<td>464.71</td>
<td>1038.7</td>
<td>478.25</td>
</tr>
<tr>
<td>Median</td>
<td>315.84</td>
<td>319.12</td>
<td>306.80</td>
<td>746.27</td>
<td>173.99</td>
</tr>
<tr>
<td>Sharp</td>
<td>2645.77</td>
<td>2522.62</td>
<td>2543.3</td>
<td>2872.66</td>
<td>3279.17</td>
</tr>
<tr>
<td>Blure</td>
<td>515.03</td>
<td>447.81</td>
<td>424.23</td>
<td>905.98</td>
<td>674.43</td>
</tr>
<tr>
<td>Emboss</td>
<td>12491.3</td>
<td>4893.4</td>
<td>4739.17</td>
<td>7326.78</td>
<td>10820.3</td>
</tr>
<tr>
<td>Oilpaint</td>
<td>761.22</td>
<td>531.59</td>
<td>568.37</td>
<td>917.52</td>
<td>617.77</td>
</tr>
<tr>
<td>$\overline{\sigma}$</td>
<td>136.95</td>
<td>200.64</td>
<td>196.46</td>
<td>537.93</td>
<td>110.09</td>
</tr>
<tr>
<td>$\sigma_c$</td>
<td>45.91</td>
<td>20.61</td>
<td>19.58</td>
<td>161.20</td>
<td>16.50</td>
</tr>
<tr>
<td>$\overline{\sigma}$</td>
<td>2844.41</td>
<td>1365.34</td>
<td>1339.12</td>
<td>2074.47</td>
<td>2333.15</td>
</tr>
<tr>
<td>$\sigma_o$</td>
<td>4368.73</td>
<td>1742.24</td>
<td>1697.82</td>
<td>2436.48</td>
<td>3892.41</td>
</tr>
<tr>
<td>Range[a,b]</td>
<td>[0-245.30]</td>
<td>[0-249.28]</td>
<td>[0-242.67]</td>
<td>[0-915.14]</td>
<td>[0-148.70]</td>
</tr>
<tr>
<td>$\beta$</td>
<td>0.28</td>
<td>0.26</td>
<td>0.26</td>
<td>0.32</td>
<td>0.29</td>
</tr>
</tbody>
</table>

Table 4.4: ED on 5 different starting code books
Chapter 5

Critical Set Based Image Hashing

5.1 Introduction

The basic idea is to find critical co-ordinates that capture the main features of an image and remain essentially unchanged through acceptable transformations. Here, *essentially unchanged* means the hash value is considered *close*. For $A$, the set of admissible transformations, we only consider JPEG compression with different quality levels. All other transformations, including filtering, are considered inadmissible manipulations that must be detectable by the system. However the proposed systems are flexible and can be used with other definitions of $A$. The larger transformation set means wider band for acceptance and hence higher rate of false acceptance.

The critical co-ordinates are found by taking the difference between the original image pixels and the corresponding low frequency (low-low pass) image pixels, obtained by applying a wavelet/discrete cosine transform, and keeping the co-ordinates of those differences which are above a certain threshold. By doing this we essentially capture the high frequency edge co-ordinates of an image. To be more precise in selecting the final critical set, we take a range of images in an admissible transformation and use the intersection of all those critical sets as the final critical set in the hash value. Since edge detection schemes also capture the high frequency co-ordinates as edge pixels (critical points), to compare the efficiency of our wavelet/discrete cosine based scheme with the others, we choose Canny’s 53 and XV’s 55 edge detectors to find the critical set. To make a fair comparison, we kept the size of the critical set to a constant value by choosing the other parameters appropriately. Sections 5.1.1, and 5.1.2 give a detail description of wavelet transform 52 and discrete cosine transform respectively. Section 5.1.3 describes XV’s and Canny’s edge detectors. Section 5.2 explains our scheme and Section 5.3 gives the experimental results supporting our claim. Sections 5.4 and 5.5 analyse the security level of the scheme and Section 5.6 compares our scheme with the others. Section 5.7 concludes the chapter.
5.1.1 Wavelet Transform

Wavelet transform (WT) represents an image as a sum of wavelet functions with different locations and scales. Any decomposition of an image into wavelets involves a pair of waveforms: one to represent the high frequencies corresponding to the detailed parts of an image (wavelet function $\Psi$) and one for the low frequencies or smooth parts of an image (scaling function $\Phi$).

Wavelet function $\Psi$ is used to represent detailed parts of the image and the scaling function $\Phi$ is used to represent smooth parts of the image. The two waveforms are translated and scaled on the time axis to produce a set of wavelet functions at different locations and on different scales. Each wavelet contains the same number of cycles, such that, as the frequency reduces the wavelet gets longer. High frequencies are transformed with short functions (low scale). Low frequencies are transformed with long functions (high scale). During computation, the analyzing wavelet is shifted over the full domain of the function. The result of WT is a set of wavelet coefficients, which measure the contribution of the wavelets at these locations and scales.

Multiresolution Analysis

WT performs multiresolution image analysis. The result of multiresolution analysis is simultaneous image representation on different resolution levels. The resolution is determined by a threshold below which all fluctuations or details are ignored. The difference between two neighboring resolutions represents details. Therefore, an image can be represented by a low-resolution image (approximation or average part) and the details on each higher resolution level. Let us consider a one-dimensional (1-D) function $f(t)$. At the resolution level $j$, the approximation of the function $f(t)$ is $f_j(t)$. At the next resolution level $j+1$, the approximation of the function $f(t)$ is $f_{j+1}(t)$. The details denoted by $d_j(t)$ are included in $f_{j+1}(t)$: $f_{j+1}(t) = f_j(t) + d_j(t)$. This procedure can be repeated several times and function $f(t)$ can be viewed as

$$f(t) = f_j(t) + \sum_{k=j}^{n} d_k(t)$$

Similarly, the space of square integrable functions $L^2(R)$ can be viewed as a composition of scaling subspaces $V_j$ and wavelet subspaces $W_j$ such that the approximation of $f(t)$ at resolution $j(f_j(t))$ is in $V_j$ and the details $d_j(t)$ are in $W_j$. $V_j$ and $W_j$ are defined in terms of dilates and translates of scaling function $\Phi$ and wavelet function $\Psi$: $V_j = \{\Phi(2^j x - k)|k \in Z\}$ and $W_j = \{\Psi(2^j x - k)|k \in Z\}$. $V_j$ and $W_j$ are localized in dyadically scaled frequency "octaves" by the scale or resolution parameter $2^j$ and
localized spatially by translation $k$. The scaling subspace $V_j$ must be contained in all subspaces on higher resolutions ($V_j \subset V_{j+1}$). The wavelet subspaces $W_j$ fill the gaps between successive scales: $V_{j+1} = V_j \oplus W_j$. We can start with an approximation on some scale $V_0$ then use wavelets to fill in the missing details on finer and finer scales. The finest resolution level includes all square integrable functions

$$L^2(R) = V_0 + \bigoplus_{j=0}^{\infty} W_j$$

Since $\Phi \in V_0 \subset V_1$, it follows that the scaling function for multiresolution approximation can be obtained as the solution to a two-scale dilation equation

$$\Phi(x) = \sum_k a_L(k)\Phi(2x - k)$$

for some suitable sequence of coefficients $a_L(k)$. Once $\Phi$ has been found, an associated mother wavelet is given by a similar looking formula

$$\Psi(x) = \sum_k a_H(k)\Phi(2x - k)$$

Some effort is required to produce appropriate coefficients sequences $a_L(k)$ and $a_H(k)$.

**Discrete Wavelet Transform**

One of the big discoveries for wavelet analysis was that perfect reconstruction filter banks could be formed using the coefficient sequences $a_L(k)$ and $a_H(k)$. The input sequence is convolved with high-pass (HPF) and low-pass (LPF) filters $a_H(k)$ and $a_L(k)$ and each result is downsampled by two, yielding the transformed signals $x_H$ and $x_L$. The signal is reconstructed through upsampling and convolving with high and low synthesis filters $s_H(k)$ and $s_L(k)$. For properly designed filters, the signal $x$ is reconstructed exactly ($y = x$).

The choice of filter not only determines whether perfect reconstruction is possible, it also determines the shape of wavelet we use to perform the analysis. By cascading the analysis filter bank with itself a number of times, a digital signal decomposition with dyadic frequency scaling known as DWT can be formed. The mathematical manipulation that effects synthesis is called inverse DWT. An efficient way to implement this scheme using filters was developed by Mallat [54]. The new twist that wavelets bring to filter banks is connection between multiresolution analysis (that, in principle, can be performed on the original, continuous signal) and digital signal processing performed on discrete, sampled signals.
DWT for an image as a 2-D signal can be derived from 1-D DWT. The easiest way for obtaining scaling and wavelet function for two dimensions is by multiplying two 1-D functions. The scaling function for 2-D DWT can be obtained by multiplying two 1-D scaling functions: \( \Phi(x, y) = \Phi(x)\Phi(y) \). Wavelet functions for 2-D DWT can be obtained by multiplying two wavelet functions or wavelet and scaling function for 1-D analysis. For the 2-D case, there exist three wavelet functions that scan details in horizontal \( \Psi^{(I)}(x, y) = \Phi(x)\Psi(y) \), vertical \( \Psi^{(II)}(x, y) = \Psi(x)\Phi(y) \), and diagonal directions: \( \Psi^{(III)}(x, y) = \Psi(x)\Psi(y) \). This may be represented as a four-channel perfect reconstruction filter bank. Now, each filter is 2-D with the subscript indicating the type of filter (HPF or LPF) for separable horizontal and vertical components. The resulting four transform components consist of all possible combination of high and low pass filtering in the two directions. By using these filters in one stage, an image can be decomposed into four bands. There are three types of detail images for each resolution: Horizontal (HL), vertical (LH), and diagonal (HH). The operations can be repeated on the low-low band using the second stage of identical filter bank. Thus, a typical 2-D DWT, used in image compression, will generate the hierarchical pyramidal structure.

Properties of Wavelets

Wavelet functions are characterized by the following properties.

1. **Support** is the finite interval over which the basis functions have been defined. Compact support is good for efficient implementation and good localization.

2. **Symmetry** of basis functions. This is desirable for linear phasing, that is to avoid dephasing.

3. **Orthogonality** of basis functions. It means the basis functions are orthonormal to each other. This property leads to faster algorithms.

4. **Regularity or degree of smoothness** is related to how many times the basis functions are differentiable.

5. **Order of filter or vanishing moments** determines the number of oscillations in the basis functions. This in turn determines the size of filter.
**Choice of Wavelet**

In our experiments we use Daubechies Wavelet (DB) and Bi-orthogonal Wavelet (BW) due to the remarkable results they produce in image analysis for the reasons described below. Each wavelet is parameterized by an integer N that determine the filter order. Bi-orthogonal wavelet can use filters with similar or dissimilar orders for decomposition (Nd) and reconstruction (Nr). We have used the following wavelets: DW-N with N=3,6, and 10 and BW-Nr-Nd with (4,4).

Daubechies wavelets are families of orthogonal wavelets that are compactly supported with more continuous derivatives and zero integrals. Compactly supported wavelets correspond to finite impulse response (FIR) filters and thus leads to efficient implementation. Only filters with infinite duration can provide perfect interband decorrelation of coefficients. Since time localization is important in visual signal processing, long filters can not be used. The major disadvantage of DW for compression is their asymmetry, which can cause artifacts at borders of the wavelet subbands. This is not a concern for hash value generation. Symmetry in wavelet can be obtained only if compact support or orthogonality is given up. Biorthogonal wavelets are compactly supported symmetric non-orthogonal wavelets.

**Filter Order and Filter Length**

The filter length L is determined by filter order, but relationship between filter order and filter length is different for different wavelet families. Higher filter orders give wider functions in the time domain with higher degree of smoothness. Filters with a higher order provide good frequency localization, which increases energy compaction. Filters with lower order have a better time localization and preserves important edge information. Wavelet based image compression prefers high degree of smooth functions, where as wavelet based hash function prefers low degree of smooth functions.

**5.1.2 Discrete Cosine Transform**

Current standards for compression of still images and moving images use DCT which represents an image as a superposition of cosine functions with different discrete frequencies. The transformed signal is a function of two spatial dimensions, and its components are called DCT coefficients or spatial frequencies. DCT coefficients measure the contribution of the cosine functions at different discrete frequencies. DCT has been recommended by JPEG because of its excellent energy compaction property.
5.1.3 Edge Detection

The edge detection we used for our experiments are Canny's [53] and XV's [55] ones.

XV Edge Detector

XV is an image editing software [55] which includes an edge detector which runs a convolution using a pair of convolutions, one which detects horizontal edges, and one which detects vertical edges. The two edge detection masks used (vertical and horizontal) simultaneously, takes the Max(abs(results)) as its gradient. The convolution masks which are used are given below.

\[
\begin{pmatrix}
-1 & 0 & 1 \\
-1 & 0 & 1 \\
-1 & 0 & 1 \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
-1 & -1 & -1 \\
0 & 0 & 0 \\
1 & 1 & 1 \\
\end{pmatrix}
\]

Table 5.1: The horizontal and vertical convolution masks for edge detection

Canny’s Edge Detector

The Canny operator [53] was designed to be an optimal edge detector (according to particular criteria — there are other detectors around that also claim to be optimal with respect to slightly different criteria). It takes as input a gray scale image, and produces as output an image showing the positions of tracked intensity discontinuities.

The Canny operator works in a multi-stage process. First of all the image is smoothed by Gaussian convolution with standard deviation \( \sigma \). Then a simple 2-D first derivative operator (somewhat like the Roberts Cross) is applied to the smoothed image to highlight regions of the image with high first spatial derivatives. Edges give rise to ridges in the gradient magnitude image. The algorithm then tracks along the top of these ridges and sets to zero all pixels that are not actually on the ridge top so as to give a thin line in the output, a process known as non-maximal suppression.

The tracking process exhibits hysteresis controlled by two thresholds: \( T_1 \) and \( T_2 \), with \( T_1 > T_2 \). Tracking can only begin at a point on a ridge higher than \( T_1 \). Tracking then continues in both directions out from that point until the height of the ridge falls below \( T_2 \). This hysteresis helps to ensure that noisy edges are not broken up into multiple edge fragments. We parameterize the whole Canny edge detection process by \( \text{Canny}(\sigma,T_1,T_2) \).
5.2 Image Hash

5.2.1 Basic Description of the Scheme

This method of hashing is based on finding the significant co-ordinates, which are considered at a range of acceptable compression levels. We call this set of points a critical set $S_C$ for DCT based JPEG compression and keep this set as one of the parameters of the hash value. By keeping this critical set we not only avoid insignificant co-ordinates, but also reduce the length of hash. This critical set will be well estimated if many compression levels are considered.

5.2.2 Hash Generation Algorithm

1. Given an image $I$, apply a wavelet transform with out subsampling or DCT transform to produce the low-low pass image $I_{LL}$. In the case of DCT, apply DCT to each block of the image, make high frequency coefficients from $AC_j$ to $AC_{63}$ zero, and apply inverse DCT to obtain the low pass image $I_{LL}$.

2. Let $\rho$ be an appropriately chosen threshold. (To be described later)

3. Compare the corresponding pixel values of $I$ and $I_{LL}$ to determine the set of all pixel co-ordinates whose differences are above $\rho$. Let us denote this set of pixel co-ordinates as $S_0$

4. Assuming compression level of $l_n$ to be used as the threshold to distinguish compression from other manipulations, JPEG compress image $I$ at quality levels $l_1$, $l_2, ..., l_n$ ($l_1 > l_2 > ... > l_n$) to produce the compressed images $I_1, I_2, ..., I_n$.

5. Apply steps (1)-(3) on compressed images $I_1, I_2, ..., I_n$ to produce the set of pixel co-ordinates $S_1, S_2, ..., S_n$, using the same $\rho$.

6. Let us define a critical set $S_C = S_0 \cap S_1 \cap S_2 \cap ... \cap S_n$. Higher the number of compression levels $n$ are considered, better the critical set $S_C$ is approximated.

7. A confidence interval must be set on the number of matching co-ordinates for authentication.

8. Hash of the image will consist of wavelet basis used, $S_C$, $\rho$, and t-a confidence measure.
5.2.3 Verification Algorithm

1. Upon receiving an image $I^r$ and its hash, apply the wavelet transform without subsampling or DCT to produce the low-low pass image $I_{LL}^r$. In the case of DCT, apply DCT to each block of the image, make high frequency co-efficients from $AC_j$ to $AC_{63}$ zero, and apply inverse DCT to obtain the low pass image $I_{LL}^r$.

2. Compare the corresponding pixel values of $I^r$ and $I_{LL}^r$ to determine the set of all pixel co-ordinates whose differences are above the received $\rho$. Let us denote this set of pixel co-ordinates as $S_r$.

3. If the number of matching co-ordinates in $S_c$ and $S_r$ is within the interval $[|S_c| - t, |S_c| + t]$, the received image is considered to be authentic.

5.2.4 Length of Hash: Measure of Security

For a hash function $h(x) = y$ to be secure, it must satisfy all collision resistant properties [50] given below:

1. The argument $x$ can be of arbitrary length and the result $h(x)$ has a fixed length of $n$ bits. (with $n \geq 128,...,160$)

2. The hash function must be one way, i.e., preimage resistance and second preimage resistance. (Given $y$ it is hard to find a message $x$ such that $h(x) = y$. (preimage resistance)
   Given $x$ and $h(x)$, it is hard to find a message $x' \neq x$ such that $h(x') = h(x)$ (second preimage resistance))

3. The hash function must be collision resistant: This means that it is hard to find two distinct messages that hash to the same result.

It has been stated in [50] that for a hash function to be secure against random preimage attack and birthday attack, the size of hash value $n$ should be at least 128...160 bits. That is, the size of hash value plays a crucial part in determining the security of a hash function. In our proposed wavelet/DCT based hash function, we are fortunate enough in setting the size of hash value by varying the threshold $\rho$ between 0 and 255. The higher the value of $\rho$, the smaller the size of critical set and the lower the value of $\rho$, the larger the size of critical set. Assuming $x$ to be a text message rather than an Image data and each critical point co-ordinate is represented as a 16-bit word, to achieve security against pre-image attack and birthday attack, there
needs to be minimum of 8 critical co-ordinates in the hash value or critical set. Hence, 
ρ’s that produce less than 8 critical points should be avoided.

5.3 Experimental Results

We performed experiments on Lena, Peppers, Pills, Paper, and Corrosion images to verify the above described scheme. The chosen compression levels for finding the critical set were 75, 50, and 25 (l₁ = 75, l₂ = 50, l₃ = 25 and n = 3) assuming compression at quality level of 25 (l₃) had been used as the threshold to distinguish compression from other modifications.

The experiments can be grouped into two categories: Those that aim at optimizing parameters of the proposed scheme and those that are designed to justify the structural choices. The hash generation algorithm uses wavelet decomposition of an image and constructs low pass filtered version of the image to find the critical points. The following experiments were designed to investigate the effect of the choice of the wavelet and the choice of scale in determining the critical set.

5.3.1 Constant Value for ρ within One Scale

Objective and Experiment

The objective of this experiment was to see how the number of pixels in the critical set varies when ρ was fixed for different types of filters (DW-3, DW-6, DW-10, and BO-(4,4)), and to evaluate the matching percentages for each manipulation. We chose ρ = 100 for all images. We chose Daubechies and bi-orthogonal wavelets for our experiments to compare the performance. For ρ = 100, the number of significant co-ordinates in the critical set Sₓ and the percentage of matching co-ordinates |Sₓ ∩ Sᵧ|/|Sₓ| * 100 for each manipulation had also been computed. The following observations were noted. (Tables 5.2-5.6)

Observations

The number of co-ordinates in the critical set increased with an increase in the size of Daubechies filter. Bi-orthogonal wavelet produced the highest number of co-ordinates in the critical set. The matching co-ordinates for images compressed at quality levels 80, 60, and 40 had always been close to 100 percent. The matching co-ordinates for manipulated images increased with an increase in vanishing moment or filter length. Bi-orthogonal wavelet had the highest matching percentage for manipulated images, thus
reducing the deviation from acceptable compression levels. Daubechies wavelet with the least vanishing moment or filter length produced the least matching percentages for manipulated images, thus increasing the deviation from acceptable compression levels.

<table>
<thead>
<tr>
<th>Image</th>
<th>DB-3</th>
<th>DB-6</th>
<th>DB-10</th>
<th>BO-(4,4)</th>
<th>XV</th>
</tr>
</thead>
<tbody>
<tr>
<td>(</td>
<td>S_c</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\rho</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Lena.10</td>
<td>65%</td>
<td>80%</td>
<td>83%</td>
<td>82%</td>
<td>87%</td>
</tr>
<tr>
<td>Lena.FMLR</td>
<td>62%</td>
<td>79%</td>
<td>75%</td>
<td>83%</td>
<td>91%</td>
</tr>
<tr>
<td>Lena.Gauss</td>
<td>18%</td>
<td>74%</td>
<td>81%</td>
<td>91%</td>
<td>98%</td>
</tr>
<tr>
<td>Lena.Median</td>
<td>58%</td>
<td>76%</td>
<td>80%</td>
<td>80%</td>
<td>81%</td>
</tr>
<tr>
<td>Lena.80</td>
<td>100%</td>
<td>98%</td>
<td>98%</td>
<td>99%</td>
<td>99%</td>
</tr>
<tr>
<td>Lena.60</td>
<td>97%</td>
<td>97%</td>
<td>98%</td>
<td>98%</td>
<td>99%</td>
</tr>
<tr>
<td>Lena.40</td>
<td>96%</td>
<td>97%</td>
<td>96%</td>
<td>97%</td>
<td>97%</td>
</tr>
</tbody>
</table>

Table 5.2: The number of co-ordinates in a critical set and the matching percentages on modified images for wavelets DB-3, DB-6, DB-10, and BO-(4,4), and XV’s edge detector

<table>
<thead>
<tr>
<th>Image</th>
<th>DB-3</th>
<th>DB-6</th>
<th>DB-10</th>
<th>BO-(4,4)</th>
<th>XV</th>
</tr>
</thead>
<tbody>
<tr>
<td>(</td>
<td>S_c</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\rho</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Peppers.10</td>
<td>38%</td>
<td>62%</td>
<td>79%</td>
<td>84%</td>
<td>92%</td>
</tr>
<tr>
<td>Peppers.FMLR</td>
<td>67%</td>
<td>54%</td>
<td>68%</td>
<td>88%</td>
<td>95%</td>
</tr>
<tr>
<td>Peppers.Gauss</td>
<td>5%</td>
<td>34%</td>
<td>70%</td>
<td>94%</td>
<td>99%</td>
</tr>
<tr>
<td>Peppers.Median</td>
<td>71%</td>
<td>63%</td>
<td>78%</td>
<td>91%</td>
<td>81%</td>
</tr>
<tr>
<td>Peppers.80</td>
<td>95%</td>
<td>97%</td>
<td>98%</td>
<td>99%</td>
<td>99%</td>
</tr>
<tr>
<td>Peppers.60</td>
<td>95%</td>
<td>96%</td>
<td>98%</td>
<td>99%</td>
<td>99%</td>
</tr>
<tr>
<td>Peppers.40</td>
<td>86%</td>
<td>94%</td>
<td>95%</td>
<td>99%</td>
<td>99%</td>
</tr>
</tbody>
</table>

Table 5.3: The number of co-ordinates in a critical set and the matching percentages on modified images for wavelets DB-3, DB-6, DB-10, and BO-(4,4), and XV’s edge detector
<table>
<thead>
<tr>
<th>Image</th>
<th>DB-3</th>
<th>DB-6</th>
<th>DB-10</th>
<th>BO-(4,4)</th>
<th>XV</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>S_c</td>
<td>$</td>
<td>213</td>
<td>863</td>
<td>1718</td>
</tr>
<tr>
<td>$\rho$</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Pills.10</td>
<td>44%</td>
<td>77%</td>
<td>83%</td>
<td>87%</td>
<td>74%</td>
</tr>
<tr>
<td>Pills.FMLR</td>
<td>81%</td>
<td>88%</td>
<td>89%</td>
<td>92%</td>
<td>91%</td>
</tr>
<tr>
<td>Pills.Gauss</td>
<td>7%</td>
<td>55%</td>
<td>81%</td>
<td>90%</td>
<td>98%</td>
</tr>
<tr>
<td>Pills.Median</td>
<td>79%</td>
<td>89%</td>
<td>92%</td>
<td>92%</td>
<td>87%</td>
</tr>
<tr>
<td>Pills.80</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Pills.60</td>
<td>95%</td>
<td>94%</td>
<td>95%</td>
<td>97%</td>
<td>97%</td>
</tr>
<tr>
<td>Pills.40</td>
<td>96%</td>
<td>97%</td>
<td>98%</td>
<td>98%</td>
<td>96%</td>
</tr>
</tbody>
</table>

Table 5.4: The number of co-ordinates in a critical set and the matching percentages on modified images for wavelets DB-3, DB-6, DB-10, and BO-(4,4), and XV’s edge detector

<table>
<thead>
<tr>
<th>Image</th>
<th>DB-3</th>
<th>DB-6</th>
<th>DB-10</th>
<th>BO-(4,4)</th>
<th>XV</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>S_c</td>
<td>$</td>
<td>15</td>
<td>127</td>
<td>468</td>
</tr>
<tr>
<td>$\rho$</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Paper.10</td>
<td>47%</td>
<td>51%</td>
<td>69%</td>
<td>73%</td>
<td>63%</td>
</tr>
<tr>
<td>Paper.FMLR</td>
<td>80%</td>
<td>76%</td>
<td>69%</td>
<td>77%</td>
<td>75%</td>
</tr>
<tr>
<td>Paper.Gauss</td>
<td>40%</td>
<td>39%</td>
<td>59%</td>
<td>59%</td>
<td>88%</td>
</tr>
<tr>
<td>Paper.Median</td>
<td>80%</td>
<td>81%</td>
<td>88%</td>
<td>80%</td>
<td>70%</td>
</tr>
<tr>
<td>Paper.80</td>
<td>100%</td>
<td>96%</td>
<td>96%</td>
<td>98%</td>
<td>99%</td>
</tr>
<tr>
<td>Paper.60</td>
<td>100%</td>
<td>96%</td>
<td>96%</td>
<td>98%</td>
<td>97%</td>
</tr>
<tr>
<td>Paper.40</td>
<td>93%</td>
<td>86%</td>
<td>92%</td>
<td>95%</td>
<td>96%</td>
</tr>
</tbody>
</table>

Table 5.5: The number of co-ordinates in a critical set and the matching percentages on modified images for wavelets DB-3, DB-6, DB-10, and BO-(4,4), and XV’s edge detector
<table>
<thead>
<tr>
<th>Image</th>
<th>DB-3</th>
<th>DB-6</th>
<th>DB-10</th>
<th>BO-(4,4)</th>
<th>XV</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>S_c</td>
<td>$</td>
<td>16</td>
<td>89</td>
<td>199</td>
</tr>
<tr>
<td>$\rho$</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Corrosion.10</td>
<td>69%</td>
<td>60%</td>
<td>62%</td>
<td>76%</td>
<td>88%</td>
</tr>
<tr>
<td>Corrosion.FMLR</td>
<td>44%</td>
<td>75%</td>
<td>82%</td>
<td>78%</td>
<td>93%</td>
</tr>
<tr>
<td>Corrosion.Gauss</td>
<td>0%</td>
<td>18%</td>
<td>40%</td>
<td>65%</td>
<td>96%</td>
</tr>
<tr>
<td>Corrosion.Median</td>
<td>56%</td>
<td>47%</td>
<td>48%</td>
<td>72%</td>
<td>79%</td>
</tr>
<tr>
<td>Corrosion.80</td>
<td>87%</td>
<td>97%</td>
<td>96%</td>
<td>96%</td>
<td>98%</td>
</tr>
<tr>
<td>Corrosion.60</td>
<td>87%</td>
<td>98%</td>
<td>98%</td>
<td>95%</td>
<td>99%</td>
</tr>
<tr>
<td>Corrosion.40</td>
<td>81%</td>
<td>90%</td>
<td>92%</td>
<td>93%</td>
<td>98%</td>
</tr>
</tbody>
</table>

Table 5.6: The number of co-ordinates in a critical set and the matching percentages on modified images for wavelets DB-3, DB-6, DB-10, and BO-(4,4), and XV’s edge detector.
5.3.2 Constant Value for $\rho$ within Scale One and Two

Objective and Experiment

The objective of this experiment was to evaluate the effect of considering more than one scale in determining the critical set. Instead of only comparing the pixel values of the image $I$ with the corresponding ones on the low-low pass version of the image ($I_{LL}$), we also considered the difference between pixel values of $I_{LL}$ and $I_{LLLL}$ which was the low-low pass version of $I_{LL}$. The following observations were noted. (Tables 5.7-5.11)

Observations

Incorporating two levels of scale decreased the number of co-ordinates in the critical set. The number of co-ordinates in the critical set increased with an increase in the size of Daubechies filter. Bi-orthogonal wavelet produced the highest number of co-ordinates in the critical set except for Paper image. The matching co-ordinates for images compressed at quality levels 80, 60, and 40 had been close to 100 percent except for Corrosion image. The matching co-ordinates for manipulated images did not increase with an increase in vanishing moments for some modifications (FMLR, Gaussian, and Median filtering). Bi-orthogonal wavelet had the highest matching percentage for manipulated images, thus reducing the deviation from acceptable compression levels. Daubechies wavelet with the least vanishing moment or filter length did not produce the least matching percentages for manipulated images on all occasions. On Corrosion, Pills and Paper images some manipulations were not distinguished from compression. This indicated that going for more than one scale deteriorated performance due to the finer representation of the critical set.
<table>
<thead>
<tr>
<th>Image</th>
<th>DB-3</th>
<th>DB-6</th>
<th>DB-10</th>
<th>BO-(4,4)</th>
<th>XV</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>S_c</td>
<td>$</td>
<td>25</td>
<td>199</td>
<td>470</td>
</tr>
<tr>
<td>$\rho$</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Lena.10</td>
<td>12%</td>
<td>74%</td>
<td>83%</td>
<td>84%</td>
<td>87%</td>
</tr>
<tr>
<td>Lena.FMLR</td>
<td>68%</td>
<td>89%</td>
<td>86%</td>
<td>91%</td>
<td>91%</td>
</tr>
<tr>
<td>Lena.Gauss</td>
<td>8%</td>
<td>84%</td>
<td>92%</td>
<td>94%</td>
<td>98%</td>
</tr>
<tr>
<td>Lena.Median</td>
<td>52%</td>
<td>68%</td>
<td>78%</td>
<td>86%</td>
<td>81%</td>
</tr>
<tr>
<td>Lena.80</td>
<td>96%</td>
<td>98%</td>
<td>98%</td>
<td>99%</td>
<td>99%</td>
</tr>
<tr>
<td>Lena.60</td>
<td>100%</td>
<td>98%</td>
<td>98%</td>
<td>98%</td>
<td>99%</td>
</tr>
<tr>
<td>Lena.40</td>
<td>84%</td>
<td>95%</td>
<td>96%</td>
<td>98%</td>
<td>97%</td>
</tr>
</tbody>
</table>

Table 5.7: The number of co-ordinates in a critical set and the matching percentages on modified images for wavelets DB-3, DB-6, DB-10, and BO-(4,4), and XV's edge detector

<table>
<thead>
<tr>
<th>Image</th>
<th>DB-3</th>
<th>DB-6</th>
<th>DB-10</th>
<th>BO-(4,4)</th>
<th>XV</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>S_c</td>
<td>$</td>
<td>7</td>
<td>135</td>
<td>530</td>
</tr>
<tr>
<td>$\rho$</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Peppers.10</td>
<td>0%</td>
<td>63%</td>
<td>78%</td>
<td>85%</td>
<td>92%</td>
</tr>
<tr>
<td>Peppers.FMLR</td>
<td>86%</td>
<td>70%</td>
<td>79%</td>
<td>90%</td>
<td>95%</td>
</tr>
<tr>
<td>Peppers.Gauss</td>
<td>0%</td>
<td>85%</td>
<td>88%</td>
<td>96%</td>
<td>99%</td>
</tr>
<tr>
<td>Peppers.Median</td>
<td>71%</td>
<td>53%</td>
<td>71%</td>
<td>90%</td>
<td>81%</td>
</tr>
<tr>
<td>Peppers.80</td>
<td>100%</td>
<td>98%</td>
<td>99%</td>
<td>99%</td>
<td>99%</td>
</tr>
<tr>
<td>Peppers.60</td>
<td>100%</td>
<td>96%</td>
<td>99%</td>
<td>99%</td>
<td>99%</td>
</tr>
<tr>
<td>Peppers.40</td>
<td>100%</td>
<td>92%</td>
<td>96%</td>
<td>98%</td>
<td>99%</td>
</tr>
</tbody>
</table>

Table 5.8: The number of co-ordinates in a critical set and the matching percentages on modified images for wavelets DB-3, DB-6, DB-10, and BO-(4,4), and XV's edge detector
### Table 5.9: The number of co-ordinates in a critical set and the matching percentages on modified images for wavelets DB-3, DB-6, DB-10, and BO-(4,4), and XV’s edge detector

<table>
<thead>
<tr>
<th>Image</th>
<th>DB-3</th>
<th>DB-6</th>
<th>DB-10</th>
<th>BO-(4,4)</th>
<th>XV</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>S_e</td>
<td>_\rho$</td>
<td>50</td>
<td>421</td>
<td>628</td>
</tr>
<tr>
<td>Pills.10</td>
<td>68%</td>
<td>72%</td>
<td>85%</td>
<td>84%</td>
<td>74%</td>
</tr>
<tr>
<td>Pills.FMLR</td>
<td>*86%</td>
<td>*95%</td>
<td>*98%</td>
<td>94%</td>
<td>91%</td>
</tr>
<tr>
<td>Pills.Gauss</td>
<td>64%</td>
<td>78%</td>
<td>86%</td>
<td>89%</td>
<td>98%</td>
</tr>
<tr>
<td>Pills.Median</td>
<td>74%</td>
<td>80%</td>
<td>85%</td>
<td>94%</td>
<td>87%</td>
</tr>
<tr>
<td>Pills.80</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Pills.60</td>
<td>84%</td>
<td>98%</td>
<td>97%</td>
<td>97%</td>
<td>97%</td>
</tr>
<tr>
<td>Pills.40</td>
<td>96%</td>
<td>92%</td>
<td>97%</td>
<td>98%</td>
<td>96%</td>
</tr>
</tbody>
</table>

### Table 5.10: The number of co-ordinates in a critical set and the matching percentages on modified images for wavelets DB-3, DB-6, DB-10, and BO-(4,4), and XV’s edge detector

<table>
<thead>
<tr>
<th>Image</th>
<th>DB-3</th>
<th>DB-6</th>
<th>DB-10</th>
<th>BO-(4,4)</th>
<th>XV</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>S_e</td>
<td>_\rho$</td>
<td>14</td>
<td>264</td>
<td>341</td>
</tr>
<tr>
<td>Paper.10</td>
<td>50%</td>
<td>77%</td>
<td>84%</td>
<td>49%</td>
<td>63%</td>
</tr>
<tr>
<td>Paper.FMLR</td>
<td>86%</td>
<td>91%</td>
<td>79%</td>
<td>90%</td>
<td>75%</td>
</tr>
<tr>
<td>Paper.Gauss</td>
<td>50%</td>
<td>65%</td>
<td>74%</td>
<td>46%</td>
<td>88%</td>
</tr>
<tr>
<td>Paper.Median</td>
<td>*93%</td>
<td>67%</td>
<td>73%</td>
<td>64%</td>
<td>70%</td>
</tr>
<tr>
<td>Paper.80</td>
<td>93%</td>
<td>99%</td>
<td>99%</td>
<td>96%</td>
<td>99%</td>
</tr>
<tr>
<td>Paper.60</td>
<td>100%</td>
<td>97%</td>
<td>95%</td>
<td>96%</td>
<td>97%</td>
</tr>
<tr>
<td>Paper.40</td>
<td>93%</td>
<td>96%</td>
<td>95%</td>
<td>91%</td>
<td>96%</td>
</tr>
<tr>
<td>Image</td>
<td>DB-3</td>
<td>DB-6</td>
<td>DB-10</td>
<td>BO-(4,4)</td>
<td>XV</td>
</tr>
<tr>
<td>---------------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
<td>----------</td>
<td>-----</td>
</tr>
<tr>
<td>$</td>
<td>S_c</td>
<td>$</td>
<td>10</td>
<td>173</td>
<td>372</td>
</tr>
<tr>
<td>$\rho$</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Corrosion.10</td>
<td>50%</td>
<td>72%</td>
<td>80%</td>
<td>86%</td>
<td>88%</td>
</tr>
<tr>
<td>Corrosion.FMLR</td>
<td>*80%</td>
<td>94%</td>
<td>94%</td>
<td>*97%</td>
<td>93%</td>
</tr>
<tr>
<td>Corrosion.Gauss</td>
<td>*80%</td>
<td>*77%</td>
<td>91%</td>
<td>*92%</td>
<td>96%</td>
</tr>
<tr>
<td>Corrosion.Median</td>
<td>50%</td>
<td>*72%</td>
<td>87%</td>
<td>87%</td>
<td>79%</td>
</tr>
<tr>
<td>Corrosion.80</td>
<td>70%</td>
<td>77%</td>
<td>96%</td>
<td>97%</td>
<td>98%</td>
</tr>
<tr>
<td>Corrosion.60</td>
<td>100%</td>
<td>98%</td>
<td>97%</td>
<td>99%</td>
<td>99%</td>
</tr>
<tr>
<td>Corrosion.40</td>
<td>100%</td>
<td>81%</td>
<td>95%</td>
<td>92%</td>
<td>98%</td>
</tr>
</tbody>
</table>

Table 5.11: The number of co-ordinates in a critical set and the matching percentages on modified images for wavelets DB-3, DB-6, DB-10, and BO-(4,4), and XV’s edge detector.
5.3.3 A Comparison with Edge Detection

Objective and Experiment

The objective of this experiment was to compare the performance of our hash function with similar schemes such as edge detection algorithms which find the pixels that correspond to high frequency components of the image. We used Canny’s and XV’s edge detectors to find these points. The points that were obtained through the comparison of an image \( I \) and its low-low pass version \( I_{LL} \) correspond to high frequency components of the image.

We performed experiments using XV’s edge detector after thresholding the edge pixel with \( \rho = 100 \), and Canny’s edge detector to compare the performance with the wavelet based ones. Since Canny gave only a binary image as its output we decided to use only the edge pixels for our comparisons. We also decided to fix the parameters used for hysteresis since we believed that fixing these parameters wouldn’t affect the critical edge pixels very much. Only the standard deviation of the Gaussian convolution was varied from 1 to 3. (ie. \( \sigma = 1, 2, \text{ and } 3 \)). We observed the following. (Tables 5.12-5.16)

Observations

Gaussian filtering had not been detected by XV’s edge detector on all images. The performance of XV’s edge detector was far inferior to wavelet based ones. With an increase in standard deviation of the Gaussian kernel used in Canny, the matching percentages for manipulated images also increased, thus reducing the distinction between compression and other manipulations. This might be due to increase in \( \sigma \) that reduces the sensitivity to noise or variations. Although the number of points in the critical set decreased with an increase in \( \sigma \) for Canny based method, the number of co-ordinates in the critical set were much more than the wavelet based scheme with \( \rho = 100 \). Canny’s performance was far better than the edge detector used in XV for small values of \( \sigma \). Daubechies wavelet with the least vanishing moment or filter length discriminated other manipulations from JPEG compression far better than Canny’s and XV’s edge detector for the parameters used above.
### Table 5.12: The number of co-ordinates in a critical set and the matching percentages on modified images for Canny

<table>
<thead>
<tr>
<th>Image</th>
<th>Canny (1,0.5,0.5)</th>
<th>Canny (2,0.5,0.5)</th>
<th>Canny (3,0.5,0.5)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(</td>
<td>S_c</td>
<td>)</td>
</tr>
<tr>
<td>Lena.10</td>
<td>69%</td>
<td>82%</td>
<td>88%</td>
</tr>
<tr>
<td>Lena.FMLR</td>
<td>81%</td>
<td>78%</td>
<td>80%</td>
</tr>
<tr>
<td>Lena.Gauss</td>
<td>88%</td>
<td>93%</td>
<td>97%</td>
</tr>
<tr>
<td>Lena.Median</td>
<td>80%</td>
<td>79%</td>
<td>80%</td>
</tr>
<tr>
<td>Lena.80</td>
<td>99%</td>
<td>99%</td>
<td>100%</td>
</tr>
<tr>
<td>Lena.60</td>
<td>98%</td>
<td>99%</td>
<td>99%</td>
</tr>
<tr>
<td>Lena.40</td>
<td>94%</td>
<td>97%</td>
<td>98%</td>
</tr>
</tbody>
</table>

### Table 5.13: The number of co-ordinates in a critical set and the matching percentages on modified images for Canny

<table>
<thead>
<tr>
<th>Image</th>
<th>Canny (1,0.5,0.5)</th>
<th>Canny (2,0.5,0.5)</th>
<th>Canny (3,0.5,0.5)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(</td>
<td>S_c</td>
<td>)</td>
</tr>
<tr>
<td>Peppers.10</td>
<td>74%</td>
<td>82%</td>
<td>87%</td>
</tr>
<tr>
<td>Peppers.FMLR</td>
<td>82%</td>
<td>80%</td>
<td>80%</td>
</tr>
<tr>
<td>Peppers.Gauss</td>
<td>91%</td>
<td>91%</td>
<td>95%</td>
</tr>
<tr>
<td>Peppers.Median</td>
<td>81%</td>
<td>78%</td>
<td>80%</td>
</tr>
<tr>
<td>Peppers.80</td>
<td>98%</td>
<td>99%</td>
<td>99%</td>
</tr>
<tr>
<td>Peppers.60</td>
<td>97%</td>
<td>99%</td>
<td>99%</td>
</tr>
<tr>
<td>Peppers.40</td>
<td>96%</td>
<td>97%</td>
<td>99%</td>
</tr>
</tbody>
</table>

### Table 5.14: The number of co-ordinates in a critical set and the matching percentages on modified images for Canny

<table>
<thead>
<tr>
<th>Image</th>
<th>Canny (1,0.5,0.5)</th>
<th>Canny (2,0.5,0.5)</th>
<th>Canny (3,0.5,0.5)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(</td>
<td>S_c</td>
<td>)</td>
</tr>
<tr>
<td>Pills.10</td>
<td>83%</td>
<td>90%</td>
<td>90%</td>
</tr>
<tr>
<td>Pills.FMLR</td>
<td>93%</td>
<td>91%</td>
<td>90%</td>
</tr>
<tr>
<td>Pills.Gauss</td>
<td>92%</td>
<td>95%</td>
<td>97%</td>
</tr>
<tr>
<td>Pills.Median</td>
<td>81%</td>
<td>83%</td>
<td>82%</td>
</tr>
<tr>
<td>Pills.80</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Pills.60</td>
<td>98%</td>
<td>99%</td>
<td>99%</td>
</tr>
<tr>
<td>Pills.40</td>
<td>98%</td>
<td>99%</td>
<td>99%</td>
</tr>
<tr>
<td>Image</td>
<td>Canny(1,0,5,0.5)</td>
<td>Canny(2,0.5,0.5)</td>
<td>Canny(3,0.5,0.5)</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------</td>
<td>------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>$</td>
<td>S_c</td>
<td>$</td>
<td>3260</td>
</tr>
<tr>
<td>Paper.10</td>
<td>83%</td>
<td>88%</td>
<td>87%</td>
</tr>
<tr>
<td>Paper.FMLR</td>
<td>93%</td>
<td>92%</td>
<td>89%</td>
</tr>
<tr>
<td>Paper.Gauss</td>
<td>85%</td>
<td>91%</td>
<td>95%</td>
</tr>
<tr>
<td>Paper.Median</td>
<td>85%</td>
<td>84%</td>
<td>96%</td>
</tr>
<tr>
<td>Paper.80</td>
<td>99%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Paper.60</td>
<td>98%</td>
<td>99%</td>
<td>99%</td>
</tr>
<tr>
<td>Paper.40</td>
<td>97%</td>
<td>98%</td>
<td>98%</td>
</tr>
</tbody>
</table>

Table 5.15: The number of co-ordinates in a critical set and the matching percentages on modified images for Canny

<table>
<thead>
<tr>
<th>Image</th>
<th>Canny(1,0,5,0.5)</th>
<th>Canny(2,0.5,0.5)</th>
<th>Canny(3,0.5,0.5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>S_c</td>
<td>$</td>
<td>3260</td>
</tr>
<tr>
<td>Corrosion.10</td>
<td>65%</td>
<td>76%</td>
<td>78%</td>
</tr>
<tr>
<td>Corrosion.FMLR</td>
<td>93%</td>
<td>89%</td>
<td>91%</td>
</tr>
<tr>
<td>Corrosion.Gauss</td>
<td>86%</td>
<td>90%</td>
<td>92%</td>
</tr>
<tr>
<td>Corrosion.Median</td>
<td>85%</td>
<td>81%</td>
<td>82%</td>
</tr>
<tr>
<td>Corrosion.80</td>
<td>98%</td>
<td>99%</td>
<td>99%</td>
</tr>
<tr>
<td>Corrosion.60</td>
<td>96%</td>
<td>97%</td>
<td>98%</td>
</tr>
<tr>
<td>Corrosion.40</td>
<td>93%</td>
<td>95%</td>
<td>96%</td>
</tr>
</tbody>
</table>

Table 5.16: The number of co-ordinates in a critical set and the matching percentages on modified images for Canny
5.3.4 Fixed Size Hash

Objective and Experiment

The objective of this experiment was to evaluate the performance of the hash function when the size of critical set had been made roughly equal. We considered different wavelets (DW-3, DW-6, DW-10, and BO-(4,4)), DCT based systems, and edge based system (Canny’s). In the case of wavelets and DCT the values of $\rho$ were chosen to result in approximately the same size hash. For Canny the standard deviation of the Gaussian convolution was changed while keeping the values of $T_1$ and $T_2$ to be equal to 0.5. Although this was not the best method for comparing the efficiency of hash functions based on two different methods, we believed this might give some fair comparisons. An experiment was performed using discrete cosine transform to compare the efficiency of wavelet by keeping the size of the hash to be roughly equal to the others. (Tables 5.17-5.21)

Observations

For fixed size critical sets, Daubechies with least vanishing moment or filter length didn’t always produce the least matching percentage for all manipulations except compression. This appeared to be contrary to the results we obtained for fixed size $\rho$, where Daubechies wavelet with the least vanishing moment or filter length produced the least matching percentages for manipulated images. This revealed that by varying $\rho$, wavelets other than Daubechies could be used to optimise detection. We also observed that DCT’s (ie. when DC – AC5 coefficients had been used in smoothing) performance was slightly better than the wavelets that we used.
### Table 5.17: The number of co-ordinates in a critical set and the matching percentages on modified images for wavelets DB-3, DB-6, DB-10, and BO-(4,4), Canny’s edge detectors, and DCT

<table>
<thead>
<tr>
<th>Image</th>
<th>DB-3</th>
<th>DB-6</th>
<th>DB-10</th>
<th>BO-(4,4)</th>
<th>Canny-(9,0.5,0.5)</th>
<th>DCT</th>
</tr>
</thead>
<tbody>
<tr>
<td>(</td>
<td>S_c</td>
<td>)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\rho)</td>
<td>915</td>
<td>922</td>
<td>811</td>
<td>837</td>
<td>939</td>
<td>888</td>
</tr>
<tr>
<td>Lena.10</td>
<td>69%</td>
<td>79%</td>
<td>84%</td>
<td>78%</td>
<td>86%</td>
<td>60%</td>
</tr>
<tr>
<td>Lena.FMLR</td>
<td>82%</td>
<td>77%</td>
<td>75%</td>
<td>75%</td>
<td>86%</td>
<td>75%</td>
</tr>
<tr>
<td>Lena.Gauss</td>
<td>66%</td>
<td>73%</td>
<td>76%</td>
<td>83%</td>
<td>92%</td>
<td>74%</td>
</tr>
<tr>
<td>Lena.Median</td>
<td>71%</td>
<td>77%</td>
<td>80%</td>
<td>72%</td>
<td>80%</td>
<td>74%</td>
</tr>
<tr>
<td>Lena.80</td>
<td>97%</td>
<td>98%</td>
<td>97%</td>
<td>98%</td>
<td>98%</td>
<td>96%</td>
</tr>
<tr>
<td>Lena.60</td>
<td>97%</td>
<td>97%</td>
<td>97%</td>
<td>96%</td>
<td>96%</td>
<td>95%</td>
</tr>
<tr>
<td>Lena.40</td>
<td>94%</td>
<td>94%</td>
<td>95%</td>
<td>97%</td>
<td>95%</td>
<td>91%</td>
</tr>
</tbody>
</table>

### Table 5.18: The number of co-ordinates in a critical set and the matching percentages on modified images for wavelets DB-3, DB-6, DB-10, and BO-(4,4), Canny’s edge detector, and DCT

<table>
<thead>
<tr>
<th>Image</th>
<th>DB-3</th>
<th>DB-6</th>
<th>DB-10</th>
<th>BO-(4,4)</th>
<th>Canny-(8,0.5,0.5)</th>
<th>DCT</th>
</tr>
</thead>
<tbody>
<tr>
<td>(</td>
<td>S_c</td>
<td>)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\rho)</td>
<td>1075</td>
<td>1135</td>
<td>1114</td>
<td>1244</td>
<td>1046</td>
<td>1193</td>
</tr>
<tr>
<td>Peppers.10</td>
<td>68%</td>
<td>79%</td>
<td>83%</td>
<td>81%</td>
<td>91%</td>
<td>64%</td>
</tr>
<tr>
<td>Peppers.FMLR</td>
<td>82%</td>
<td>77%</td>
<td>74%</td>
<td>82%</td>
<td>88%</td>
<td>76%</td>
</tr>
<tr>
<td>Peppers.Gauss</td>
<td>70%</td>
<td>77%</td>
<td>79%</td>
<td>90%</td>
<td>94%</td>
<td>54%</td>
</tr>
<tr>
<td>Peppers.Median</td>
<td>76%</td>
<td>78%</td>
<td>81%</td>
<td>90%</td>
<td>70%</td>
<td>76%</td>
</tr>
<tr>
<td>Peppers.80</td>
<td>98%</td>
<td>99%</td>
<td>99%</td>
<td>99%</td>
<td>99%</td>
<td>97%</td>
</tr>
<tr>
<td>Peppers.60</td>
<td>97%</td>
<td>98%</td>
<td>99%</td>
<td>99%</td>
<td>98%</td>
<td>96%</td>
</tr>
<tr>
<td>Peppers.40</td>
<td>95%</td>
<td>96%</td>
<td>96%</td>
<td>96%</td>
<td>98%</td>
<td>93%</td>
</tr>
<tr>
<td>Image</td>
<td>DB-3</td>
<td>DB-6</td>
<td>DB-10</td>
<td>BO-(4,4)</td>
<td>Canny-(8,0,5,0,5)</td>
<td>DCT</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
<td>----------</td>
<td>------------------</td>
<td>-----</td>
</tr>
<tr>
<td>$</td>
<td>S_c</td>
<td>$</td>
<td>898</td>
<td>863</td>
<td>897</td>
<td>851</td>
</tr>
<tr>
<td>$\rho$</td>
<td>70</td>
<td>100</td>
<td>120</td>
<td>145</td>
<td></td>
<td>38</td>
</tr>
<tr>
<td>Pills.10</td>
<td>63%</td>
<td>77%</td>
<td>80%</td>
<td>82%</td>
<td>85%</td>
<td>61%</td>
</tr>
<tr>
<td>Pills.FMLR</td>
<td>88%</td>
<td>88%</td>
<td>87%</td>
<td>83%</td>
<td>93%</td>
<td>83%</td>
</tr>
<tr>
<td>Pills.Gauss</td>
<td>49%</td>
<td>55%</td>
<td>71%</td>
<td>82%</td>
<td>88%</td>
<td>41%</td>
</tr>
<tr>
<td>Pilss.Median</td>
<td>87%</td>
<td>89%</td>
<td>91%</td>
<td>90%</td>
<td>85%</td>
<td>73%</td>
</tr>
<tr>
<td>Pills.80</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>96%</td>
<td>100%</td>
</tr>
<tr>
<td>Pills.60</td>
<td>96%</td>
<td>94%</td>
<td>94%</td>
<td>95%</td>
<td>95%</td>
<td>93%</td>
</tr>
<tr>
<td>Pilss.40</td>
<td>96%</td>
<td>97%</td>
<td>97%</td>
<td>96%</td>
<td>97%</td>
<td>95%</td>
</tr>
</tbody>
</table>

Table 5.19: The number of co-ordinates in a critical set and the matching percentages on modified images for wavelets DB-3, DB-6, DB-10, and BO-(4,4), Canny’s edge detector and DCT

<table>
<thead>
<tr>
<th>Image</th>
<th>DB-3</th>
<th>DB-6</th>
<th>DB-10</th>
<th>BO-(4,4)</th>
<th>Canny-(9,5,0,5,0,5)</th>
<th>DCT</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>S_c</td>
<td>$</td>
<td>791</td>
<td>756</td>
<td>738</td>
<td>744</td>
</tr>
<tr>
<td>$\rho$</td>
<td>50</td>
<td>75</td>
<td>93</td>
<td>115</td>
<td></td>
<td>45</td>
</tr>
<tr>
<td>Paper.10</td>
<td>53%</td>
<td>62%</td>
<td>70%</td>
<td>71%</td>
<td>79%</td>
<td>67%</td>
</tr>
<tr>
<td>Paper.FMLR</td>
<td>83%</td>
<td>81%</td>
<td>72%</td>
<td>69%</td>
<td>89%</td>
<td>82%</td>
</tr>
<tr>
<td>Paper.Gauss</td>
<td>55%</td>
<td>54%</td>
<td>64%</td>
<td>56%</td>
<td>91%</td>
<td>29%</td>
</tr>
<tr>
<td>Paper.Median</td>
<td>82%</td>
<td>85%</td>
<td>89%</td>
<td>78%</td>
<td>69%</td>
<td>74%</td>
</tr>
<tr>
<td>Paper.80</td>
<td>98%</td>
<td>97%</td>
<td>96%</td>
<td>98%</td>
<td>98%</td>
<td>98%</td>
</tr>
<tr>
<td>Paper.60</td>
<td>97%</td>
<td>97%</td>
<td>95%</td>
<td>96%</td>
<td>95%</td>
<td>97%</td>
</tr>
<tr>
<td>Paper.40</td>
<td>90%</td>
<td>93%</td>
<td>95%</td>
<td>94%</td>
<td>97%</td>
<td>94%</td>
</tr>
</tbody>
</table>

Table 5.20: The number of co-ordinates in a critical set and the matching percentages on modified images for wavelets DB-3, DB-6, DB-10, and BO-(4,4), Canny’s edge detector, and DCT
<table>
<thead>
<tr>
<th>Image</th>
<th>DB-3</th>
<th>DB-6</th>
<th>DB-10</th>
<th>BO-(4,4)</th>
<th>Canny-(10,0.5,0.5)</th>
<th>DCT</th>
</tr>
</thead>
<tbody>
<tr>
<td>(</td>
<td>S_c</td>
<td>)</td>
<td>578</td>
<td>535</td>
<td>583</td>
<td>572</td>
</tr>
<tr>
<td>( \rho )</td>
<td>50</td>
<td>70</td>
<td>80</td>
<td>95</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>Corrosion.10</td>
<td>56%</td>
<td>60%</td>
<td>66%</td>
<td>76%</td>
<td>81%</td>
<td>55%</td>
</tr>
<tr>
<td>Corrosion.FMLR</td>
<td>83%</td>
<td>82%</td>
<td>84%</td>
<td>80%</td>
<td>90%</td>
<td>80%</td>
</tr>
<tr>
<td>Corrosion.Gauss</td>
<td>36%</td>
<td>43%</td>
<td>59%</td>
<td>65%</td>
<td>89%</td>
<td>24%</td>
</tr>
<tr>
<td>Corrosion.Median</td>
<td>64%</td>
<td>60%</td>
<td>57%</td>
<td>70%</td>
<td>82%</td>
<td>64%</td>
</tr>
<tr>
<td>Corrosion.80</td>
<td>95%</td>
<td>97%</td>
<td>96%</td>
<td>95%</td>
<td>94%</td>
<td>96%</td>
</tr>
<tr>
<td>Corrosion.60</td>
<td>94%</td>
<td>96%</td>
<td>97%</td>
<td>95%</td>
<td>94%</td>
<td>94%</td>
</tr>
<tr>
<td>Corrosion.40</td>
<td>89%</td>
<td>93%</td>
<td>94%</td>
<td>90%</td>
<td>91%</td>
<td>90%</td>
</tr>
</tbody>
</table>

Table 5.21: The number of co-ordinates in a critical set and the matching percentages on modified images for wavelets DB-3, DB-6, DB-10, and BO-(4,4), Canny's edge detector and DCT
5.3.5 Invariance of the Critical Set to Different Methods

Objective and Experiment

The objective of this experiment was to evaluate the invariance of the critical set to different methods used. That was to see if for a fixed size hash there was a subset of pixels that remain the same within the critical set when the derivation method was varied (DW-3, DW-6, DW-10, BO-(4,4), DCT, or Canny). If this set was large enough the authentication scheme could be invariant to the transform. (Tables 5.22-5.26)

Observations

Our Daubechies, biorthogonal, discrete cosine transform, and Canny based hashing schemes showed that the overlap between the critical sets were not enough to make the assumption that equal size critical sets produced using different methods of hashing would give a good comparison of the efficiency (manipulation detection) of hash functions based on different methods.

<table>
<thead>
<tr>
<th>Lenna</th>
<th>DB12 (922)</th>
<th>DB20 (811)</th>
<th>SP (837)</th>
<th>DCT (888)</th>
<th>Canny (939)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB6 (915)</td>
<td>592</td>
<td>344</td>
<td>178</td>
<td>223</td>
<td>10</td>
</tr>
<tr>
<td>DB12 (922)</td>
<td>541</td>
<td>275</td>
<td>171</td>
<td>109</td>
<td>14</td>
</tr>
<tr>
<td>DB20 (811)</td>
<td></td>
<td>382</td>
<td></td>
<td>109</td>
<td>19</td>
</tr>
<tr>
<td>SP (837)</td>
<td></td>
<td></td>
<td>171</td>
<td>109</td>
<td>19</td>
</tr>
<tr>
<td>DCT (888)</td>
<td></td>
<td></td>
<td></td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.22: The number of critical points common to different methods

<table>
<thead>
<tr>
<th>Peppers</th>
<th>DB12 (1135)</th>
<th>DB20 (1114)</th>
<th>SP (1244)</th>
<th>DCT (1193)</th>
<th>Canny (1034)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB6 (1075)</td>
<td>747</td>
<td>487</td>
<td>217</td>
<td>299</td>
<td>61</td>
</tr>
<tr>
<td>DB12 (1135)</td>
<td></td>
<td>788</td>
<td>363</td>
<td>342</td>
<td>79</td>
</tr>
<tr>
<td>DB20 (1114)</td>
<td></td>
<td></td>
<td>541</td>
<td>271</td>
<td>75</td>
</tr>
<tr>
<td>SP (1244)</td>
<td></td>
<td></td>
<td></td>
<td>136</td>
<td>140</td>
</tr>
<tr>
<td>DCT (1193)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>59</td>
</tr>
</tbody>
</table>

Table 5.23: The number of critical points common to different methods
### Table 5.24: The number of critical points common to different methods

<table>
<thead>
<tr>
<th>Paper</th>
<th>DB12 (756)</th>
<th>DB20 (738)</th>
<th>SP (744)</th>
<th>DCT (742)</th>
<th>Canny (800)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB6 (791)</td>
<td>496</td>
<td>261</td>
<td>107</td>
<td>125</td>
<td>84</td>
</tr>
<tr>
<td>DB12 (756)</td>
<td>434</td>
<td>162</td>
<td>321</td>
<td>120</td>
<td>88</td>
</tr>
<tr>
<td>DB20 (738)</td>
<td></td>
<td>109</td>
<td>212</td>
<td>71</td>
<td></td>
</tr>
<tr>
<td>SP (744)</td>
<td></td>
<td></td>
<td></td>
<td>66</td>
<td></td>
</tr>
<tr>
<td>DCT (742)</td>
<td></td>
<td></td>
<td></td>
<td>38</td>
<td></td>
</tr>
</tbody>
</table>

### Table 5.25: The number of critical points common to different methods

<table>
<thead>
<tr>
<th>Pills</th>
<th>DB12 (863)</th>
<th>DB20 (897)</th>
<th>SP (851)</th>
<th>DCT (846)</th>
<th>Canny (889)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB6 (898)</td>
<td>601</td>
<td>153</td>
<td>263</td>
<td>195</td>
<td>56</td>
</tr>
<tr>
<td>DB12 (863)</td>
<td>612</td>
<td>397</td>
<td>216</td>
<td>192</td>
<td>59</td>
</tr>
<tr>
<td>DB20 (897)</td>
<td></td>
<td></td>
<td>116</td>
<td>116</td>
<td>60</td>
</tr>
<tr>
<td>SP (851)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>54</td>
</tr>
<tr>
<td>DCT (846)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>40</td>
</tr>
</tbody>
</table>

### Table 5.26: The number of critical points common to different methods

<table>
<thead>
<tr>
<th>Corrosion</th>
<th>DB12 (535)</th>
<th>DB20 (583)</th>
<th>SP (572)</th>
<th>DCT (546)</th>
<th>Canny (564)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB6 (578)</td>
<td>344</td>
<td>115</td>
<td>161</td>
<td>261</td>
<td>10</td>
</tr>
<tr>
<td>DB12 (535)</td>
<td>210</td>
<td>143</td>
<td>146</td>
<td>138</td>
<td>10</td>
</tr>
<tr>
<td>DB20 (583)</td>
<td>331</td>
<td>146</td>
<td>138</td>
<td>93</td>
<td>15</td>
</tr>
<tr>
<td>SP (572)</td>
<td></td>
<td></td>
<td>93</td>
<td></td>
<td>13</td>
</tr>
<tr>
<td>DCT (546)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9</td>
</tr>
</tbody>
</table>
Figure 5.1: Critical co-ordinates for images (a) Lena (DB-3, DB-6, DB-10, BO-(4,4), DCT) (b) Peppers (DB-3, DB-6, DB-10, BO-(4,4), DCT) (c) Pills (DB-3, DB-6, DB-10, BO-(4,4), DCT) (d) Paper (DB-3, DB-6, DB-10, BO-(4,4), DCT) and (e) Corrosion (DB-3, DB-6, DB-10, BO-(4,4), DCT)
5.3.6 Optimizing AC Coefficient Addition

Objective and Experiment

As noted before the basic approach in deriving the critical set was to find a smooth version of the image. In DCT based approach the smooth version could be obtained by considering a subset, for example DC together with $AC_1$ and $AC_2$, and use the inverse transform to construct the smooth version. The number of coefficients that were used to construct the smooth version was investigated in this experiment. AC coefficients were diagonally incorporated (i.e. DC-AC5, DC-AC9, ...). We kept the threshold $\rho$ at a fixed value of 27, and evaluated the size of critical set as the AC coefficients were being added.

Observations

The higher the number of low frequency AC coefficients incorporated in smoothing an image, the lesser the number of critical points found in the critical set. This was because when more AC coefficients were considered, the low pass image became less smooth (close to original) and hence the number of pixel differences above a certain threshold decreased. This led to reduced number of points in the critical set. Next we kept the size of the hash roughly equal, and evaluated the performance of manipulation detection with respect to various combinations of AC and DC (i.e. DC-AC5, DC-AC9, ...). Tables 5.27-5.31 show the results.

Incorporating more number of AC coefficients in smoothing did improve manipulation detection for modifications such as compression at quality level of 10, and Gaussian noise addition. Performance deteriorated for FMLR with the addition of AC coefficients. Mixed results were found for Median filtering.

As we kept on adding AC coefficients, critical points found at the compression level of 25 decreased due to the closeness of compressed image and the low passed version of the compressed image.

Since the compressed image had already under gone a low pass filtering, slight low pass filtering again wouldn’t pick many high frequency points in the critical set. This reduced the number of points in the critical set as more and more AC coefficients were being added even if the threshold had been lowered.

Since we included compression quality level of 25 in our critical set computation to give enough tolerance for compression, the property mentioned above limited us from going for higher number of AC coefficients being added if we wanted to have a reasonable size critical set.
From these experiments we found that for a fixed size hash, there was a trade off between the level of compression and the number of AC coefficients considered in smoothing the image. For a fixed level of compression tolerance, adding more number of AC coefficients decreased the size of hash. This in turn reduced the manipulation detection.

For a reliable system there must be a compromise between the following:

- The size of the hash or critical set,
- Number of AC coefficients used in smoothing,
- Compression tolerance level.

The experiment we did on Lenna and Peppers images revealed that adding 44% of the AC coefficients diagonally from left to right decreased the size of critical set to 596 (from 738) and 541 (from 1291) even though the threshold had been reduced to 1. Almost a similar observation was found for Pills image. For Paper and Corrosion images, adding 56% of the AC coefficients diagonally from left to right decreased the size of critical set from their original sizes.

This showed the optimal percentages of AC coefficients that gave the highest level of detection while keeping the size of the critical set roughly equal were 33% for Lenna, Peppers and Pills images, and 44% for Paper and Corrosion images.

The above analysis explained the limitations of incorporating more and more AC coefficients in smoothing an image in our critical set based hash function surviving acceptable level of compression. We found that the optimum percentage of AC coefficients needed to maximize the manipulation detection for an equal size hash were about 33% to Lenna, Peppers and Pills images and about 44% to Paper and Corrosion images.
<table>
<thead>
<tr>
<th>DCT Coefficients</th>
<th>DC-AC5</th>
<th>DC-AC9</th>
<th>DC-AC14</th>
<th>DC-AC20</th>
<th>DC-AC27</th>
</tr>
</thead>
<tbody>
<tr>
<td>% of AC Coeff. added</td>
<td>9%</td>
<td>16%</td>
<td>23%</td>
<td>33%</td>
<td>44%</td>
</tr>
<tr>
<td>Critical set</td>
<td>888</td>
<td>790</td>
<td>798</td>
<td>738</td>
<td>596</td>
</tr>
<tr>
<td>$\rho$</td>
<td>27</td>
<td>19</td>
<td>13</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>Comp10</td>
<td>60%</td>
<td>41%</td>
<td>30%</td>
<td>4%</td>
<td>6%</td>
</tr>
<tr>
<td>FMLR</td>
<td>75%</td>
<td>79%</td>
<td>78%</td>
<td>79%</td>
<td></td>
</tr>
<tr>
<td>Gaussian</td>
<td>39%</td>
<td>34%</td>
<td>29%</td>
<td>24%</td>
<td></td>
</tr>
<tr>
<td>Median</td>
<td>74%</td>
<td>72%</td>
<td>70%</td>
<td>72%</td>
<td></td>
</tr>
<tr>
<td>Comp80</td>
<td>96%</td>
<td>96%</td>
<td>93%</td>
<td>92%</td>
<td></td>
</tr>
<tr>
<td>Comp60</td>
<td>95%</td>
<td>94%</td>
<td>92%</td>
<td>91%</td>
<td></td>
</tr>
<tr>
<td>Comp40</td>
<td>91%</td>
<td>88%</td>
<td>84%</td>
<td>85%</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.27: Matching percentage for modified images of Lena

<table>
<thead>
<tr>
<th>DCT Coefficients</th>
<th>DC-AC5</th>
<th>DC-AC9</th>
<th>DC-AC14</th>
<th>DC-AC20</th>
<th>DC-AC27</th>
</tr>
</thead>
<tbody>
<tr>
<td>% of AC Coeff. added</td>
<td>9%</td>
<td>16%</td>
<td>23%</td>
<td>33%</td>
<td>44%</td>
</tr>
<tr>
<td>Critical set</td>
<td>1193</td>
<td>1203</td>
<td>1221</td>
<td>1238</td>
<td>541</td>
</tr>
<tr>
<td>$\rho$</td>
<td>24</td>
<td>16</td>
<td>9</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Comp10</td>
<td>64%</td>
<td>47%</td>
<td>27%</td>
<td>8%</td>
<td>24%</td>
</tr>
<tr>
<td>FMLR</td>
<td>76%</td>
<td>77%</td>
<td>79%</td>
<td>86%</td>
<td></td>
</tr>
<tr>
<td>Gaussian</td>
<td>54%</td>
<td>50%</td>
<td>45%</td>
<td>66%</td>
<td></td>
</tr>
<tr>
<td>Median</td>
<td>76%</td>
<td>73%</td>
<td>74%</td>
<td>84%</td>
<td></td>
</tr>
<tr>
<td>Comp80</td>
<td>97%</td>
<td>95%</td>
<td>94%</td>
<td>91%</td>
<td></td>
</tr>
<tr>
<td>Comp60</td>
<td>96%</td>
<td>95%</td>
<td>93%</td>
<td>92%</td>
<td></td>
</tr>
<tr>
<td>Comp40</td>
<td>93%</td>
<td>91%</td>
<td>86%</td>
<td>89%</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.28: Matching percentage for modified images of Peppers

<table>
<thead>
<tr>
<th>DCT Coefficients</th>
<th>DC-AC5</th>
<th>DC-AC9</th>
<th>DC-AC14</th>
<th>DC-AC20</th>
<th>DC-AC27</th>
</tr>
</thead>
<tbody>
<tr>
<td>% of AC Coeff. added</td>
<td>9%</td>
<td>16%</td>
<td>23%</td>
<td>33%</td>
<td>44%</td>
</tr>
<tr>
<td>Critical set</td>
<td>883</td>
<td>869</td>
<td>872</td>
<td>934</td>
<td></td>
</tr>
<tr>
<td>$\rho$</td>
<td>38</td>
<td>27</td>
<td>17</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Comp10</td>
<td>61%</td>
<td>48%</td>
<td>31%</td>
<td>2%</td>
<td></td>
</tr>
<tr>
<td>FMLR</td>
<td>83%</td>
<td>82%</td>
<td>84%</td>
<td>85%</td>
<td></td>
</tr>
<tr>
<td>Gaussian</td>
<td>41%</td>
<td>27%</td>
<td>21%</td>
<td>18%</td>
<td></td>
</tr>
<tr>
<td>Median</td>
<td>73%</td>
<td>70%</td>
<td>67%</td>
<td>68%</td>
<td></td>
</tr>
<tr>
<td>Comp80</td>
<td>100%</td>
<td>99%</td>
<td>100%</td>
<td>99%</td>
<td></td>
</tr>
<tr>
<td>Comp60</td>
<td>93%</td>
<td>90%</td>
<td>86%</td>
<td>85%</td>
<td></td>
</tr>
<tr>
<td>Comp40</td>
<td>95%</td>
<td>93%</td>
<td>92%</td>
<td>87%</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.29: Matching percentage for modified images of Pills
<table>
<thead>
<tr>
<th>DCT Coefficients</th>
<th>DC-AC5</th>
<th>DC-AC9</th>
<th>DC-AC14</th>
<th>DC-AC20</th>
<th>DC-AC27</th>
</tr>
</thead>
<tbody>
<tr>
<td>% of AC Coeff. added</td>
<td>9%</td>
<td>16%</td>
<td>23%</td>
<td>33%</td>
<td>44%</td>
</tr>
<tr>
<td>Critical set</td>
<td>742</td>
<td>767</td>
<td>717</td>
<td>726</td>
<td>731</td>
</tr>
<tr>
<td>ρ</td>
<td>45</td>
<td>34</td>
<td>23</td>
<td>15</td>
<td>7</td>
</tr>
</tbody>
</table>

| Comp10           | 67%    | 57%    | 41%     | 24%     | 0%      |
| FMLR             | 82%    | 83%    | 88%     | 86%     | 92%     |
| Gaussian         | 29%    | 15%    | 9%      | 5%      | 9%      |
| Median           | 74%    | 68%    | 71%     | 75%     | 84%     |

| Comp80           | 98%    | 97%    | 97%     | 95%     | 93%     |
| Comp60           | 97%    | 95%    | 94%     | 92%     | 91%     |
| Comp40           | 94%    | 90%    | 87%     | 82%     | 86%     |

Table 5.30: Matching percentage for modified images of Paper

<table>
<thead>
<tr>
<th>DCT Coefficients</th>
<th>DC-AC5</th>
<th>DC-AC9</th>
<th>DC-AC14</th>
<th>DC-AC20</th>
<th>DC-AC27</th>
</tr>
</thead>
<tbody>
<tr>
<td>% of AC Coeff. added</td>
<td>9%</td>
<td>16%</td>
<td>23%</td>
<td>33%</td>
<td>44%</td>
</tr>
<tr>
<td>Critical set</td>
<td>546</td>
<td>500</td>
<td>523</td>
<td>484</td>
<td></td>
</tr>
<tr>
<td>ρ</td>
<td>36</td>
<td>27</td>
<td>19</td>
<td>12</td>
<td></td>
</tr>
</tbody>
</table>

| Comp10           | 55%    | 40%    | 25%     | 5%      |         |
| FMLR             | 80%    | 78%    | 80%     | 81%     |         |
| Gaussian         | 24%    | 13%    | 9%      | 10%     |         |
| Median           | 64%    | 60%    | 58%     | 61%     |         |

| Comp80           | 96%    | 93%    | 91%     | 90%     |         |
| Comp60           | 94%    | 94%    | 92%     | 93%     |         |
| Comp40           | 90%    | 89%    | 85%     | 79%     |         |

Table 5.31: Matching percentage for modified images of Corrosion
5.3.7 Performance in High-High Pass Band

Objective and Experiment

Instead of finding the pixels corresponding to the high frequency components of an image, we could directly use the output of the high-high pass filter. The objective of this experiment was to evaluate the performance of our hash function if only high-high pass band was used. We decided to evaluate the performance of our wavelet based hash function surviving acceptable level of compression in the high-high pass band after thresholding the high-high pass image with varying $\rho$s. In the case of low-low pass band, we took the difference between the original and low-low pass image and thresholded the differences to find the critical set. Here, we thresholded the high-high pass band image to compute the critical set. We chose $\rho = 105$ for this experiment. Daubechie with filter lengths of 6, 12, and 20 had been used for this $\rho$.

Observations

The detection performance was good with compression quality level of 10, and Median convolutions. On all images, FMLR attack didn’t show much deviation from compression in the high-high pass band. Gaussian noise addition was not detected on some images for Daubechie wavelet with lengths 6 and 12. We also observed that the pixel values were clustered around certain values. Only a few coefficients were below these values, thus making our critical set to be very small. Since a critical set with a few feature points would not capture enough information, the hashing scheme based on high-high pass band might not be efficient enough in detecting manipulations other than compression as our previous scheme. (Tables 5.32-5.36)

<table>
<thead>
<tr>
<th>Wavelet Critical set</th>
<th>DB3</th>
<th>DB6</th>
<th>DB12</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rho$</td>
<td>33</td>
<td>46</td>
<td>39</td>
</tr>
<tr>
<td>Comp10</td>
<td>11 (33%)</td>
<td>11 (24%)</td>
<td>14 (36%)</td>
</tr>
<tr>
<td>FMLR</td>
<td>30* (90%)</td>
<td>33 (72%)</td>
<td>26 (67%)</td>
</tr>
<tr>
<td>Gaussian</td>
<td>2 (6%)</td>
<td>22 (48%)</td>
<td>35 (90%)</td>
</tr>
<tr>
<td>Median</td>
<td>3 (9%)</td>
<td>0 (0%)</td>
<td>0 (0%)</td>
</tr>
<tr>
<td>Sharpen</td>
<td>23 (69%)</td>
<td>39* (85%)</td>
<td>35* (90%)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Wavelet Critical set</th>
<th>DB3</th>
<th>DB6</th>
<th>DB12</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rho$</td>
<td>60</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>Comp80</td>
<td>31 (93%)</td>
<td>39* (85%)</td>
<td>26* (67%)</td>
</tr>
<tr>
<td>Comp60</td>
<td>32 (97%)</td>
<td>29* (63%)</td>
<td>21* (54%)</td>
</tr>
<tr>
<td>Comp40</td>
<td>31 (94%)</td>
<td>38* (83%)</td>
<td>27* (69%)</td>
</tr>
</tbody>
</table>

Table 5.32: Matching percentage for modified images of Lenna
5.3.8 Real World Attacks

Objective

The objective of this experiment was to evaluate the performance of our hash function with respect to other real world geometric attacks as well as some image processing modifications using statistical hypothesis of testing.

Testing of Hypothesis

- Hypothesis $H_0$: Image under consideration is original or compressed to permissible quality level if the matching percentages fall within the range of $[a, b]$. 

Table 5.33: Matching percentage for modified images of Peppers

<table>
<thead>
<tr>
<th>Wavelet Critical set $\rho$</th>
<th>DB3</th>
<th>DB6</th>
<th>DB12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comp10</td>
<td>7 (13%)</td>
<td>11 (18%)</td>
<td>9 (31%)</td>
</tr>
<tr>
<td>FMLR</td>
<td>49* (91%)</td>
<td>34 (56%)</td>
<td>14 (48%)</td>
</tr>
<tr>
<td>Gaussian</td>
<td>54* (100%)</td>
<td>37 (60%)</td>
<td>3 (10%)</td>
</tr>
<tr>
<td>Median</td>
<td>5 (9%)</td>
<td>8 (13%)</td>
<td>5 (17%)</td>
</tr>
<tr>
<td>Sharpen</td>
<td>40 (74%)</td>
<td>51* (84%)</td>
<td>20 (60%)</td>
</tr>
<tr>
<td>Comp80</td>
<td>47 (87%)</td>
<td>45* (74%)</td>
<td>19* (66%)</td>
</tr>
<tr>
<td>Comp60</td>
<td>52 (96%)</td>
<td>57 (93%)</td>
<td>13* (45%)</td>
</tr>
<tr>
<td>Comp40</td>
<td>41 (76%)</td>
<td>50 (81%)</td>
<td>29 (100%)</td>
</tr>
</tbody>
</table>

Table 5.34: Matching percentage for modified images of Paper

<table>
<thead>
<tr>
<th>Wavelet Critical set $\rho$</th>
<th>DB3</th>
<th>DB6</th>
<th>DB12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comp10</td>
<td>37 (44%)</td>
<td>15 (26%)</td>
<td>20 (36%)</td>
</tr>
<tr>
<td>FMLR</td>
<td>84* (98%)</td>
<td>56* (97%)</td>
<td>46* (82%)</td>
</tr>
<tr>
<td>Gaussian</td>
<td>72 (83%)</td>
<td>58* (100%)</td>
<td>26 (46%)</td>
</tr>
<tr>
<td>Median</td>
<td>10 (12%)</td>
<td>26 (45%)</td>
<td>14 (25%)</td>
</tr>
<tr>
<td>Sharpen</td>
<td>68 (80%)</td>
<td>46 (79%)</td>
<td>39 (70%)</td>
</tr>
<tr>
<td>Comp80</td>
<td>74 (87%)</td>
<td>53 (91%)</td>
<td>40* (71%)</td>
</tr>
<tr>
<td>Comp60</td>
<td>77 (91%)</td>
<td>55 (95%)</td>
<td>50 (89%)</td>
</tr>
<tr>
<td>Comp40</td>
<td>77 (91%)</td>
<td>54 (93%)</td>
<td>52 (93%)</td>
</tr>
<tr>
<td>Wavelet Critical set</td>
<td>DB3</td>
<td>DB6</td>
<td>DB12</td>
</tr>
<tr>
<td>----------------------</td>
<td>------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>$\rho$</td>
<td>105</td>
<td>105</td>
<td>105</td>
</tr>
<tr>
<td>Comp10</td>
<td>52 (19%)</td>
<td>50 (43%)</td>
<td>38 (44%)</td>
</tr>
<tr>
<td>FMLR</td>
<td>195 (73%)</td>
<td>111* (95%)</td>
<td>86* (99%)</td>
</tr>
<tr>
<td>Gaussian</td>
<td>140 (52%)</td>
<td>18 (15%)</td>
<td>16 (18%)</td>
</tr>
<tr>
<td>Median</td>
<td>83 (31%)</td>
<td>49 (42%)</td>
<td>40 (46%)</td>
</tr>
<tr>
<td>Sharpen</td>
<td>237 (88%)</td>
<td>102 (87%)</td>
<td>68 (78%)</td>
</tr>
<tr>
<td>Comp80</td>
<td>267 (99%)</td>
<td>117 (100%)</td>
<td>87 (100%)</td>
</tr>
<tr>
<td>Comp60</td>
<td>161* (60%)</td>
<td>100* (85%)</td>
<td>76* (87%)</td>
</tr>
<tr>
<td>Comp40</td>
<td>197* (74%)</td>
<td>108 (92%)</td>
<td>73* (84%)</td>
</tr>
</tbody>
</table>

Table 5.35: Matching percentage for modified images of Pills

<table>
<thead>
<tr>
<th>Wavelet Critical set</th>
<th>DB3</th>
<th>DB6</th>
<th>DB12</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rho$</td>
<td>105</td>
<td>105</td>
<td>105</td>
</tr>
<tr>
<td>Comp10</td>
<td>68 (26%)</td>
<td>24 (34%)</td>
<td>9 (25%)</td>
</tr>
<tr>
<td>FMLR</td>
<td>203 (77%)</td>
<td>62* (87%)</td>
<td>31* (86%)</td>
</tr>
<tr>
<td>Gaussian</td>
<td>164 (63%)</td>
<td>38 (54%)</td>
<td>15 (42%)</td>
</tr>
<tr>
<td>Median</td>
<td>27 (10%)</td>
<td>11 (15%)</td>
<td>10 (28%)</td>
</tr>
<tr>
<td>Sharpen</td>
<td>183 (70%)</td>
<td>48 (68%)</td>
<td>23 (64%)</td>
</tr>
<tr>
<td>Comp80</td>
<td>178 (68%)</td>
<td>63 (89%)</td>
<td>29* (81%)</td>
</tr>
<tr>
<td>Comp60</td>
<td>226 (87%)</td>
<td>70 (99%)</td>
<td>35 (97%)</td>
</tr>
<tr>
<td>Comp40</td>
<td>218 (84%)</td>
<td>64 (90%)</td>
<td>32 (88%)</td>
</tr>
</tbody>
</table>

Table 5.36: Matching percentage for modified images of Corrosion

- Hypothesis $H_1$: Image under consideration is rejected as manipulated if the matching percentages fall out of the range of $[a,b]$.

Rejecting $H_o$ when it should be accepted is the level of significance of the test. The probability of accepting $H_o$ when actually an image is manipulated is given by area $\beta$. We chose $\alpha = 0.1$ as the level of significance of the test.

The attacks we chose for this experiment were blurring, embossing, horizontal flip, removal, oilpainting, rotation, sharpening, and vertical flip.
Figure 5.2: $\alpha$, $\beta$, and the acceptance region $[a, b]$ assuming the values follow normal distributions with means $\bar{\sigma}$ and $\bar{\sigma}$ and standard deviations $\sigma_c$ and $\sigma_o$ respectively.

**Observations**

The matching percentages for manipulated images were far less than 100%. Rotation, flipping, oilpainting, blurring, and embossing had been clearly distinguished from compression. We purposely omitted small modifications and sharpening in this statistical evaluation to give reliable results. The same manipulations were done on Lenna, Peppers, Pills, Paper and Corrosion images and averages on all five images were tabulated. (Table 5.37) We saw a clear distinction between compression and other modifications. Acceptance range had also been determined for each wavelet/DCT used. Type 2 error was always 0, which indicated the reliability of our system. All these experiments showed our compression surviving hashing scheme’s superior performance.
<table>
<thead>
<tr>
<th>Method</th>
<th>DB3</th>
<th>DB6</th>
<th>DB10</th>
<th>BO(4,4)</th>
<th>DCT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comp10</td>
<td>61.8%</td>
<td>71.4%</td>
<td>76.6%</td>
<td>77.6%</td>
<td>61.4%</td>
</tr>
<tr>
<td>FMLR</td>
<td>83.6%</td>
<td>81.0%</td>
<td>78.4%</td>
<td>77.8%</td>
<td>74.2%</td>
</tr>
<tr>
<td>Gauss</td>
<td>55.2%</td>
<td>60.4%</td>
<td>69.8%</td>
<td>75.2%</td>
<td>37.4%</td>
</tr>
<tr>
<td>Median</td>
<td>76.0%</td>
<td>77.8%</td>
<td>79.6%</td>
<td>80.0%</td>
<td>72.2%</td>
</tr>
<tr>
<td>Blur</td>
<td>37.2%</td>
<td>41.0%</td>
<td>53.6%</td>
<td>66.8%</td>
<td>20.4%</td>
</tr>
<tr>
<td>Emboss</td>
<td>13.8%</td>
<td>6.8%</td>
<td>2.0%</td>
<td>0.2%</td>
<td>3.6%</td>
</tr>
<tr>
<td>Hor-flip</td>
<td>11.6%</td>
<td>9.2%</td>
<td>8.4%</td>
<td>6.8%</td>
<td>8.6%</td>
</tr>
<tr>
<td>Oilpaint</td>
<td>27.0%</td>
<td>25.2%</td>
<td>26.8%</td>
<td>41.4%</td>
<td>33.8%</td>
</tr>
<tr>
<td>Rotate</td>
<td>37.4%</td>
<td>39%</td>
<td>44.2%</td>
<td>49.6%</td>
<td>38.6%</td>
</tr>
<tr>
<td>Ver-flip</td>
<td>12.8%</td>
<td>8.8%</td>
<td>7.6%</td>
<td>6.4%</td>
<td>9.6%</td>
</tr>
<tr>
<td>-</td>
<td>41.6%</td>
<td>42.06</td>
<td>44.7</td>
<td>48.18</td>
<td>36.48</td>
</tr>
<tr>
<td>$\sigma_x$</td>
<td>25.085</td>
<td>27.77</td>
<td>29.86</td>
<td>31.05</td>
<td>25.64</td>
</tr>
<tr>
<td>Comp80</td>
<td>97.6%</td>
<td>98.2%</td>
<td>97.6%</td>
<td>98.0%</td>
<td>97.4%</td>
</tr>
<tr>
<td>Comp60</td>
<td>96.2%</td>
<td>96.4%</td>
<td>96.4%</td>
<td>96.2%</td>
<td>95.0%</td>
</tr>
<tr>
<td>Comp40</td>
<td>92.8%</td>
<td>94.6%</td>
<td>95.4%</td>
<td>94.2%</td>
<td>92.6%</td>
</tr>
<tr>
<td>$\bar{c}$</td>
<td>93.53</td>
<td>96.4</td>
<td>96.46</td>
<td>96.3</td>
<td>95</td>
</tr>
<tr>
<td>$\sigma_c$</td>
<td>2.015</td>
<td>1.469</td>
<td>0.899</td>
<td>1.53</td>
<td>1.96</td>
</tr>
<tr>
<td>range $[a,b]$</td>
<td>[74.0,inf.]</td>
<td>[77.88,inf.]</td>
<td>[83.202,inf.]</td>
<td>[88.23,inf.]</td>
<td>[69.55,inf.]</td>
</tr>
<tr>
<td>$\beta$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 5.37: Acceptance range $[a,b]$ and $\beta$
5.4 Security Analysis

5.4.1 $\rho$ as a Key

Though keeping $\rho$ as a key is not a bad idea, it has a shortcoming. We believe the shortcomings is as follows.

1. If the algorithm to find the critical set is known or public except for the key $\rho$, it would not be difficult to guess $\rho$ from an image critical set pair.

5.4.2 Compression Levels as Keys

Keeping compression levels as keys and then choosing the $\rho$ that gives the correct size critical set will not only avoid attacks, but also protect $\rho$ from being guessed. This is because choosing different compression levels does not always lead to the same critical set. We have noticed this from some of our experiments (Table 5.38). This is also not a very good key selection because the overlap between two critical sets derived from two sets of compression levels is almost close to 100%.

Table 5.38: The number of critical points common to different combination of compression levels

<table>
<thead>
<tr>
<th>Comp. levels</th>
<th>[CS]</th>
<th>Common [CS]</th>
<th>%</th>
<th>$\rho$</th>
</tr>
</thead>
<tbody>
<tr>
<td>90, 80 and 70</td>
<td>1303</td>
<td></td>
<td>85%</td>
<td>27</td>
</tr>
<tr>
<td>80, 70 and 50</td>
<td>1139</td>
<td>1114</td>
<td>98%</td>
<td>27</td>
</tr>
<tr>
<td>90, 80, 70 and 50</td>
<td>880</td>
<td>850</td>
<td>75%</td>
<td>27</td>
</tr>
</tbody>
</table>

5.4.3 Low-pass Filter as a Key

For a fixed size critical set, two distinct filters produce more than 36% of non overlapping points. That is, by knowing a critical set produced using a filter, it is extremely hard to guess another critical set produced using a different filter. This is also evident from some experiments we have performed (Tables 5.22-5.26). Since there are numerous filters we can choose by selecting different basis functions, keeping the low pass filter (ie. filter and its properties) as a key is an attractive idea for avoiding attacks.
5.5 Size of Key Space

5.5.1 $\rho$ as a Key

Since the maximum value of pixel value difference between the original image and the lowpass image is 255, $\rho$ can take values ranging from 0 to 255. Keeping $\rho$ as 0 will include all pixel points as critical points and keeping $\rho$ as 255 will include none of the pixel points as critical points. Therefore, the key space of $\rho$ is upper bounded by the value of 255.

5.5.2 Compression Levels as Keys

JPEG compression quality level range from 0 to 100. Choosing quality levels below certain value $l$ may not produce good quality images and so lead to great proportion of insignificant critical points. Suppose $l$ has been set to an appropriate quality level (close to 100), there are $100 - l$ possible values to choose from. Since a quality level chosen to produce a set of points should not be repeated for producing another set of points, each level chosen must be distinct in the range of $l$ to 100. If $p$ (less than or equal to $100 - l + 1$) levels are chosen, the size of key space will be $(100 - l + 1)(100 - l)\ldots(100 - l - p + 2)$ which is equal to $\frac{(100 - l + 1)!}{(100 - l - p + 1)!}$

5.5.3 Low-pass Filter as a Key

In the case of DCT, there are 64 DCT coefficients available for each 8x8 block. In low pass filtering using DCT, if $m$ high frequency coefficients are made 0 uniformly among all blocks, there are $m$ possible keys. If the number of high frequency coefficients that are made 0 is varied among blocks and kept within the range of 1 to $m$, assuming $n$ blocks are there, the size of key space will be less than or equal to $m^n$.

In the case of wavelet, the number of possible low-pass wavelet coefficients are numerous. This is because the choice of basis functions are numerous. If this scheme is implemented as a part of JPEG 2000, the choice of wavelet coefficients will be restricted to the choice of wavelet coefficients used in JPEG 2000.

5.6 A Comparison

In this section we compare our scheme with the Battacharjee's [36] and Chun-Shien's [37] wavelet based authentication schemes with respect to the following:

1. Wavelet
2. Scales and size of filters
3. Computations involved
4. Threshold selection/Compression tolerance
5. Redundant information/Orientation

The above five are selected for comparison because they are not only common to our proposed hash function, but also determine the efficiency of all wavelet based hash functions proposed so far.

5.6.1 Wavelet
Battacharjee’s scheme uses Mexican-Hat wavelet which is isotropic in nature. Their reason for choosing this wavelet filter is that it detects point features. In our scheme, we have used Daubechic wavelet and bi-orthogonal wavelets which are directionally sensitive and preserve the variation in frequencies in low-low pass band better. Although Chun-Shien says that their scheme has been experimented using a wavelet transform, they haven’t mentioned which wavelet they have used.

5.6.2 Scales and Size of Filters
Bathacharjee’s scheme uses the parent child relationship between the scales which are apart (not adjacent), and they haven’t analysed the performance with respect to different size filters. Although Chun-Shien’s scheme uses the parent child relationship between adjacent scales, they haven’t looked at the parent child relationship when the scales are far apart or when different sized filters are used. Our scheme compares the wavelet co-efficients in low-low pass band within a range of adjacent scale differences using different size filters.

5.6.3 Computations Involved
Compared to Battacharjee’s scheme, the number of computations involved in our scheme is considerably reduced because

1. No local maxima for \( P_{ij}(x, \theta) \) is found around the neighborhood of \( x \).

2. No maxima is found among all orientations. (Fortunately, Mexican-hat wavelet doesn’t have a direction)
3. No variance has been calculated on the raw image.

Although our scheme is efficient in terms of performance, when computation is compared Chun-Shien’s scheme performs better with an increase in scale. This is because larger the scale, the lesser the number of wavelet coefficients. This happens because of the sub-sampling.

5.6.4 Threshold Selection/Compression Tolerance

To give provision for compression tolerance, Battcharjee’s scheme should set the thresholds $T$ and $t$ appropriately. Even if $T$ and $t$ are set, there is no guarantee that the feature co-ordinates of the original image and the compressed image will match or not. No provision for matching percentage of feature co-ordinates has been given to distinguish compression from other malicious modifications.

In our scheme, after setting the appropriate threshold $\rho$, we determine another threshold based on the matching percentage of feature co-ordinates to give provision for acceptable compression level.

Chun-Shien’s scheme does not look at the performance with respect to different $\rho$. We have experimented our scheme with different values for $\rho$ and observed the relationship between the size of hash and the value of $\rho$. There is a trade off between robustness and fragility depending on the value of $\rho$ in a certain range. We also suggest that $\rho$ should not be too large or too small. The larger the $\rho$, the shorter the size of hash and less manipulations are detected. The smaller the $\rho$, the longer the size of hash and more manipulations are detected.

5.6.5 Redundant Information/Orientation

Battcharjee’s scheme doesn’t give any provision for frequency bands within each scale, and merely compares the responses $M_i(x)$ and $M_j(x)$ between scales $i$ and $j$ in a neighbourhood around $x$. This is because they have used Mexican-hat wavelet which is not directionally sensitive in detecting point features. In our view, this is not a good selection of wavelet because it doesn’t distinguish frequency bands within each scale at all to detect JPEG compression, which is essentially a low pass filtering. It also means a lot of redundancy is involved in their scheme.

Since we use only the low-low pass band of the Daubechie and bi-orthogonal wavelets at different scales, for giving tolerance to JPEG compression which destroys the higher frequencies, our scheme will definitely perform better than theirs.
Chun-Shien’s scheme doesn’t avoid information in the higher subbands which contain irrelevant information for distinguishing JPEG compression which is a low pass filtering. We consider the parent child relationships used in the higher frequency subbands of their scheme are redundant and may give false positive results. We have observed this in some of our experiments and purposely avoided this redundancy by utilizing only the low-low pass subbands.

5.7 Conclusion

In this chapter, we have proposed a variant to already existing wavelet based image authentication scheme. An alternate discrete cosine transform based scheme has also been proposed. The use of critical set has increased performance at the cost of more computations. We have noticed that Daubechies wavelet with the vanishing moment of 3 has produced incredible results in terms of hash length and compression tolerant discrimination of manipulations for a fixed value of $\rho$. We have also compared the proposed schemes with Canny’s and XV’s edge based schemes. The experiments show that our wavelet/discrete cosine based scheme performs better than the Canny based method. Further experiments are needed using other wavelets to improve the scheme.
Chapter 6

Fragile Watermark on Critical Points

6.1 Introduction

The aim of authentication is to protect the information being transmitted against tampering. Cryptography provides the means for accomplishing the authenticity of the information being received/sent and guarantees that the information has not been tampered with. A cryptographic hash function protects the integrity of the information by ensuring that a single bit change is detectable. For images this level of protection is not so important because a change in a single pixel value will have little impact on the visual perception of the image. In fact protection against bit change is not desirable because images may undergo a range of transformations including compression and decompression which affects many bits. Watermarking as opposed to cryptographic methods provides protection, possibly localizing the tampered areas without having to append additional information. This is accomplished at the expense of slightly modifying the pixel values.

6.2 Previous Schemes

We classify previous fragile watermarking schemes into following categories: Checksum Technique, M-Sequence Approach, Two-Dimensional Spatial Watermarks, DCT Coefficient Modulation, Sub-Band Watermarking Technique, Visible Watermark, and Feature-Based Watermark. This classification has been given in [74]. We sumnerize each technique here before describing our own method of watermarking.

6.2.1 Checksum Technique

This watermark is formed from the checksum value of the seven most significant bits of all pixels [67]. A checksum is the modulo-2 addition of a sequence of fixed length binary words. In this technique, one word is the concatenation of eight 7-bit segments,
which come from eight different pixels. Each pixel is involved in the checksum only once. The final checksum is fifty-six bits. The technique then randomly chooses the locations of the pixels that are to contain one bit of the checksum. The pixel locations of the checksum, together with the checksum itself, form the watermark \( W \). The last bit of each chosen pixel is changed (if necessary) to equal the corresponding checksum bit. \( W \) must be kept secret. To verify this watermark the checksum of a test image \( Z \) is obtained and compared to the ideal version in \( W \). Any discrepancy invalidates \( Z \). The main drawback of the scheme is that an attacker could remove the entire watermark by replacing the LSB plane.

### 6.2.2 M-Sequence Approach

This watermark is based on using a modified m-sequence [75]. A linear feedback shift register with \( n \) stages can form pseudo random binary sequence with periods as large as \( 2^n - 1 \). M-sequences achieve this maximum period and have a very desirable autocorrelation and randomness properties. Two types of sequences may be formed from an m-sequence: unipolar and bipolar. The elements of a bipolar sequence are \( \{-1, 1\} \) and the elements of a unipolar sequence are \( \{0, 1\} \). In [75], \( X \) is a grayscale \( 512 \times 512 \) image, and \( W \) is an extended one-dimensional bipolar m-sequence of length \( 512 \). \( W \) consists of \( 512 \) randomly shifted copies of \( w \) - one for each row in \( X \). \( W \) is then arithmetically added to \( X \) to form the watermarked image \( Y \).

\[
Y = X + W
\]

To verify a possibly forged image row \( z \) relative to the original row \( y \), the spatial crosscorrelation function is obtained.

\[
R_{zw}(\alpha) = \sum_j [Z(j) - E(z)]w(j - \alpha)
\]

where \( E[z] \) is the average pixel value of row \( z \). The presence of the peak in \( R_{zw} \) is determined. If there is no peak, \( z \) is not authentic. The main drawback to the schemes are (1) the last two bit planes could be removed and replaced, (2) an attacker can deduce \( W \) if \( 2n \) consecutive bits in \( W \) are known.

### 6.2.3 Two Dimensional Spatial Watermarks

To localize errors in two dimensions, this method has been proposed. This can easily be achieved with using a two-dimensional watermark blocks and testing an image on a block by block basis [76] [77]. The watermark in VW2D is created as follows.
1. A bi-polar m-sequence with a period of $2^{36} - 1$ is obtained, and the first 128 bits are discarded.

2. The next 64 bits are shaped column-wise into an $8 \times 8$ block, w. The next 32 bits are discarded. This step repeats to form additional w.

$$y = x + w$$

$y$ is the watermarked block. This process is repeated until the entire image $X$ is marked. The total number of watermark blocks is image dependent; together they form the watermark $W$. To verify a possibly forged image block $z$, one must obtain the crosscorrelation function:

$$R_{xx}(\alpha, \beta) = \sum_x \sum_y z(x, y) w(x - \alpha, y - \beta)$$

$$\delta = R_{yw}(0, 0) - R_{zw}(0, 0)$$

If $\delta < T$, where $T$ is the test threshold, $z$ is genuine. Large values of $T$ tolerate changes to the marked image block $y$. If $z = y$, then $\delta = 0$. The main disadvantage to the scheme is that if an attacker knows the watermark bit $w(i, j)$ for two pixels, those two pixels can be changed by equal amounts in opposite directions without changing $\delta$.

### 6.2.4 DCT Coefficient Modulation

An algorithm that places a watermark in the most perceptually significant areas of an image is described in [78]. The marking procedure modulates the discrete cosine transform (DCT) coefficients of the image using a one-dimensional watermark $W$. The technique is robust to many types of distortion (including cropping, very low-bit rate JPEG compression and D/A conversion) as well as collusion from several independently watermarked images.

$W$ is a sequence of normally distributed, zero mean unit variance random numbers. A DCT transform is performed on the entire image and the transform coefficients are then modulated as follows. Let $X$ be the original image, $Y$ be the watermarked image and $X_D$ and $Y_D$ be the row-concatenated DCT coefficients of $X$ and $Y$, respectively. Let $X_D(i)$ and $Y_D(i)$ be the $i^{th}$ DCT coefficients in $X_D$ and $Y_D$. $W(i)$ is the $i^{th}$ element in the watermark sequence; $a$ is a scale factor which prevents unreasonable values for $Y_D(i)$. The mark is then performed:
\[ Y_D(i) = X_D(i)(1 + aW) \]

Inversely transforming \( Y_D \) to form \( Y \) completes the marking procedure. The first step of the verification procedure is to obtain a copy of \( W \) from a possibly forged watermarked image \( Z \). \( Z_D \) is the row concatenated vector of \( Z \)'s DCT coefficients. Let \( W^* \) be the extracted version of \( W \).

\[ W^*(i) = \frac{1}{a} \left[ \frac{Z_D(i)}{X_D(i)} - 1 \right] \]

A measure of similarity between \( W^* \) and the original \( W \) is defined as:

\[ S(W, W^*) = \frac{WW^*}{\sqrt{W^*W}} \]

If an image has been watermarked with \( W \), \( S \) is a zero mean random variable. If \( W^* \) differs only slightly from \( W \), then \( E(S) \gg 0 \). A hypothesis test on \( S \) can determine if \( W \) is indeed the image’s watermark. This method accommodates a much wider range of attacks.

### 6.2.5 Sub-Band Watermarking Technique

This algorithm is described in [79]. An original image is first passed through a Gabor filter bank. The energy in each band is obtained. The center frequencies of each filter is denoted as \( f_k \). \( E_k \) is the energy of the image present in the frequency band specified by filter \( k \). The next step is to obtain a random two-dimensional array which is then low-pass filtered to form \( G \). \( G \) is then modulated at each center frequency, \( f_k \) to form \( G_k \). The energy of \( G_k \), \( D_k \) is obtained. The watermark can then be formed:

\[ W = \sum_{k=1}^{n} \alpha_k W_k \]

where \( \alpha_k = 1 \) if \( E_k > D_k \) and \( \alpha_k = 0 \) if \( E_k < D_k \) Then,

\[ Y = X + W \]

The verification procedure first passes \( Y \) through the first filter bank. Then computes the crosscorrelation function between each \( Y_k \) and \( G_k \) for \( k \) such that \( \alpha_k = 1 \). From this crosscorrelation one can determine if \( G_k \) is actually present in \( Y_k \). If a sufficient number of peaks are present at a sufficient strength, the image belongs to the owner of \( W \).
6.2.6 A Visible Watermark-Young-Mintzer Scheme

IBM has developed a proprietary visible digital watermark for artwork in the Vatican library [65]. The first step in this technique is to choose a watermark $W$ (or a Logo $L$). This may be a picture of a known trademark; it does not have to be a noise like image; nor the same size as $X$. Pixels in $W$ (or $L$) are specified to be either "active" (1) or "transparent" (0). An image subblock $x$, that is the same size as $W$ is extracted. If the marked pixel is classified as transparent (0), the corresponding pixel in the image is not adjusted. For active watermark pixel (1), the image pixel’s color information is changed by a small but visible amount. The look up table or function ($f$) for producing the change is kept secret. This process is repeated until the entire image is marked. The watermark appears as a distinct visible pattern present throughout the image; the luminance plane of the image remains unaltered. Verification is performed by inspection; forgeries made in the image will erase the color adjustment and produce a visible break in the pattern. In this way the visible watermark offers forgeries and copy detection, but does not degrade the content of the image.

The above algorithm can be mathematically modeled as follows. First of all, a key dependent binary valued function $f$, $f : \{0, 1, \ldots, 255\} \rightarrow \{0, 1\}$ that maps integers from 0 to 255 to either 0 or 1, is defined. This is the look up table we mentioned before. To encode a binary logo $L$, this binary function is used. The gray scales $g(i, j)$ are modified to satisfy the following expression. $L(i, j) = f(g(i, j))$ for each pixel $(i, j)$.

There are some advantages to this scheme according to [70]. They are summarized as follows:

- Logo itself may carry some information about the creator,
- Comparing embedded logo with the recovered one, gives a visual inspection of the integrity of the image,
- Watermark is embedded somewhat deeper (+5-5 scale),
- Method is simple and fast. Hence, easy to implement

This method also is subject to attacks if same logo and key are used for more than one image. Suppose two images $I_1$ and $I_2$ of size $M \times N$ watermarked with the same key and logo $L$ produced the gray levels $g^{(1)}$ and $g^{(2)}$. We have

$$f(g^{(1)}(i, j)) = L(i, j) = f(g^{(2)}(i, j))$$

for all $(i, j)$
Using these \( M \times N \) equations 256 unknowns \( f(0), f(1), \ldots, f(255) \) can be guessed [70]. We can start with the set \( \{0, 1, \ldots, 255\} \) divided into 256 subsets, each subset having exactly one gray scale level. Then the first equation, \( f(g^{(1)}(1, 1)) = f(g^{(2)}(1, 1)) \), tells us that the values of \( f \) are the same for both \( g^{(1)}(1, 1) \) and \( g^{(2)}(1, 1) \). Thus, we can group together these two gray levels because the value of the binary function \( f_g \) is the same for both. Gradually, the 256 subsets will start to coalesce into a smaller number of larger subsets. At the end, there will be two large subsets, one corresponding to \( f = 0 \), the other to \( f = 1 \), and several other sets for which the values of \( f \) is undetermined.

Another attack, which combines portions of different images that are embedded the same logo using a similar look up table, is known as college attack. This is described in [69] and analyzed in [70]. This attack is common to all watermarking schemes.

### 6.2.7 Feature Based Watermark-Battacharjee’s Scheme

The only watermarking scheme, known to the author, which employs the notion of data features is found in [72]. This method proposes a scheme based on point features in images using a scale interaction technique based on 2D continuous wavelets. The approach for detecting feature points is as follows.

- Define the feature detector function, \( Q \) as:

\[
Q_{ij}(\vec{x}) = |M_i(\vec{x}) - \gamma M_j(\vec{x})|
\]

where \( M_i(\vec{x}) \) and \( M_j(\vec{x}) \) represent the responses of Mexican-Hat wavelets at the image location \( (\vec{x}) \) for scales \( i \) and \( j \) respectively. The normalizing constant, \( \gamma = 2^{-2(i-j)} \). The feature detector function \( Q \), is the absolute difference of the responses of the Mexican Hat wavelet applied to the image at two different scales.

- Determine points of local maxima of \( Q \). These maxima correspond to the set of potential feature points.

- Accept a point of local maximum in \( Q \) as a feature point based on a threshold.

These feature points are used to compute Voronoi partition of the image. The watermark is embedded in each segment using spread spectrum watermarking. In the recovery process the same features are detected and again used to partition the image. Then the watermark is extracted from each segment separately.
6.3 A New Fragile Watermarking Scheme

The idea of our fragile watermarking scheme come from Yeung-Mintzer’s scheme and our critical set based hash function surviving acceptable level of compression (last chapter).

6.3.1 Our Proposal

In our proposed hash function high frequency points are derived from taking the difference between the original image and the low pass filtered image and keeping the co-ordinates which are above certain threshold as the high frequency points. The critical set used for embedding the marks, is the intersection of high frequency points produced on the same image compressed at different acceptable quality levels.

A critical point can be a peak or a valley depending on whether the difference between the original pixel value and its corresponding low pass pixel value is positive or negative. If the critical point is a peak on the original image, the difference will be a positive one, and if the critical point is a valley on the image the difference will be a negative one. If the difference is a positive one increasing the pixel value at that critical point would not decrease the difference between the original pixel and the low pass pixel at that point. If the difference is a negative one decreasing the pixel value at that critical point would not decrease the difference between the original pixel and the low pass pixel at that point. The properties mentioned above will allow the critical points to be in tact during watermark recovery if the embedding does not drastically change the neighboring pixels.

We construct an algorithm preserving the above properties in the next section. Before describing the algorithm, the definition of Edit Distance [80] is given because it will be used in our algorithm for measuring the amount of change caused to the logo embedded.

**Edit Distance** $ED(n, m)$ is a measure of the similarity between two strings, which we will refer to as the source string ($O_n$) of length $n$ and the target string ($R_m$) of length $m$. The distance is the number of deletion, insertion, or substitutions required to transform $O_n$ into $R_m$. 
6.3.2 Watermark Embedding

1. Given an image $I$, apply a wavelet or DCT transform to produce the low-low pass image $I_{LL}$ without sub-sampling. In the case of DCT, apply DCT to each block of the image, make high frequency coefficients from $AC_j$ to $AC_{63}$ zero, and apply inverse DCT to obtain the low pass image $I_{LL}$.

2. By experimenting with a range of values starting from 0 to 255, choose an appropriate threshold $\rho$ which gives a critical set with the size equivalent to the size of the logo binary map. (In other words chose the minimum size critical set which is larger than the size of the logo by choosing appropriate $\rho$.

3. Compare the corresponding pixel values of $I$ and $I_{LL}$ to determine the set of all pixel co-ordinates whose differences are above $\rho$. Let us denote this set of pixel co-ordinates as $S_o$

4. Assuming compression level of $l_1$ to be used as the threshold to distinguish compression from other manipulations, JPEG compress image $I$ at quality levels $q_1, q_2, \ldots, q_n$ to produce the compressed images $I_1, I_2, \ldots, I_n$, where $q_1 < q_2 < \ldots < q_n$.

5. Apply steps (1)-(3) on compressed images $I_1, I_2, \ldots, I_n$ to produce the set of all pixel co-ordinates $S_1, S_2, \ldots, S_n$, using the same $\rho$.

6. Let us define a critical set $S_c = S_o \cap S_1 \cap \ldots \cap S_n$.

7. Let $S_{cp} = \{p_1, p_2, \ldots, p_n\}$ be the pixels corresponding to the critical set $S_c$ ordered according to the raster scan of the image, where $n$ is the size of the critical set that equals the size of the binary logo.

8. Let $S_{tp} = \{t_1, t_2, \ldots, t_n\}$ be the set corresponding to $S_{cp} = \{p_1, p_2, \ldots, p_n\}$ which classifies each critical point as a peak or a valley. That means $t_i$ takes a ”+” or a ”-” sign depending on whether the critical point is a peak or a valley. Let $O_n = \{l_1, l_2, \ldots, l_n\}$ be the raster scanned binary map of the logo.

9. Let $f : \{0, 1, \ldots, 255\} \rightarrow \{0, 1\}$ be the look up table for a particular key.

10. For each critical point from $p_1$ to $p_n$
    
    if $f(p_i) = l_i$
    
    do not modify $p_i$
else if \( t_i = "+" \)
increase \( p_i \) by a value of \( \delta \) until \( f(p_i + \delta) = l_i \)
else if \( t_i = "-" \)
decrease \( p_i \) by a value of \( \delta \) until \( f(p_i - \delta) = l_i \)

### 6.3.3 Verification Algorithm

1. Upon receiving an image \( I^r \), perform steps (1)-(6) on \( I^r \).
2. Let \( S^r_c \) be the critical set derived on the received image \( I^r \) in step 1 above.
3. Let \( S^r_p = \{ r_1, r_2, ..., r_m \} \) be the pixels corresponding to the critical set \( S^r_c \) ordered according to raster scan of the image.
4. For each critical point from \( r_1 \) to \( r_m \), determine \( f(r_i) = l_i^r \). Let \( R_m = \{ l_1^r, l_2^r, ..., l_m^r \} \) be the raster scanned binary map of the recovered logo.
5. Define the Edit Distance between \( O_n \) (original logo sequence) and \( R_m \) (recovered logo sequence) as \( ED(n, m) \).
6. if \( ED(n, m) < \tau \)
   we say that image has undergone an acceptable level of manipulation
else
   reject the image as inauthentic.
7. Visual inspection is also possible. In this case, original logo \( O_n \) must be compared to extracted logo \( R_m \). Original logo size must be known.

### 6.4 Experimental Results

#### 6.4.1 Accuracy of Critical Points Extraction

**Objective**

The objective of these experiments was to evaluate the performance of our watermarking scheme with respect to accuracy of critical point extraction. This could also be visually inspected by embedding a logo on different images and then extracting them individually.
Experiments and Observations

We performed experiments on Lena, Peppers, Pills, Paper, and Corrosion images to verify how many critical points remain intact after embedding the watermark using the algorithm described above. We also computed the PSNR values on watermarked images to measure the quality of the image after embedding. Our experiments revealed that almost 99% of the critical points were intact even after embedding on more than 2500 critical points. The PSNR value never became less than 30 even though 900 critical points were chosen for embedding on all images.

We also embedded the same logo (Figure 6.1(a)) of size 30 × 30 on all five images using a randomized look up table and appropriately chosen $\rho$s which gave the critical sets of size roughly equal to the size of the logo. (ie. with $\rho = 27, 24, 38, 45, \text{and, } 36$ for the images: Lena, Peppers, Pills, Paper, and Corrosion, respectively.) Figures 6.1(b)-(f) show the extracted logo from Lena, Peppers, Pills, Paper, and Corrosion images. On Peppers image, we found that the extracted logo had been shifted much more than the extracted logo on Lena, Pills, Paper, and Corrosion images. Unlike other images which lost only 1% of the original critical points in watermark (logo) recovery, Peppers lost 2% of the original critical points. It must have happened because of the embedding that disturbed the neighboring critical points of Peppers image. This is also evident from the Figure 6.1, which illustrates that compared to other images, critical points of peppers image are close to each other.

![Figure 6.1: (a) Original Logo (b) Logo on Lena (c) Logo on Peppers (d) Logo on Pills (e) Logo on Paper (f) Logo on Corrosion](image-url)
6.4.2 Sensitivity to Compression Levels

Objective

The objective of these experiments was to evaluate the performance of our watermarking scheme with respect to sensitivity to compression levels.

Experiments and Observations

Critical points could be computed using only the original image’s high frequency points alone or using the intersection of high frequency points of compressed images high frequency points. Using three levels of compression in determining the critical set for embedding and extraction of watermark (logo) did not make much difference. Table 6.10 and Table 6.11 show the number of critical points captured before and after embedding, and the number of critical points common to both critical sets considering 3 levels of compression and no compression at all, respectively. It also gives the PSNR values that measure the image fidelity.

6.4.3 Imperceptibility vs. Amount of Information Hidden

Imperceptibility

The modification caused by watermark embedding should be below the perceptible threshold, which means that some sort of perceptibility criterion should be used not only to design the watermark, but also to quantify the distortion. As a consequence of the required imperceptibility, the individual pixels that are used for watermark embedding are only modified by a small amount.

Size of Logo Hidden

Though our scheme visually collapses the whole watermark even if a single point is lost in the critical set, including insufficient critical points in watermark embedding will lead to manipulation without detection (false negative). Therefore, the logo embedded as watermark must be large enough to cover the entire image. This also means higher number of pixels (or critical points) must be chosen for watermark embedding.

Choosing all pixels of the image for embedding would be similar to Young-Mintzer scheme. Since Young-Mintzer scheme (Section 6.2.6) has already been broken, choosing all pixels for embedding is not preferable for security reasons. It means a sufficient number of points not equal to the size of the image must be chosen.
Imperceptibility vs. Amount of Information Hidden

Although higher imperceptibility and higher size of logo are needed to maintain the quality of the image and the security of the system, respectively, there is a trade off between these two. That is, the higher the imperceptibility, the lesser the amount of information hidden as watermark. To optimize performance, a reasonable trade off must be maintained between imperceptibility and size of logo hidden.

Objective

The objective of these experiments was to evaluate the relationship between the size of logo hidden or embedded and the amount of distortion caused by embedding.

Experiment and Observation

To show this, we embedded different size logo sequence (from 30x30 to 100x100) on Peppers image, measured the PSNR of the image and the MSE on modified pixels. Table 6.1 shows the experimental results in which one can clearly see the inverse relationship between the size of the embedded sequence and the PSNR value of the image.

<table>
<thead>
<tr>
<th>Logo size or number of embedded bits</th>
<th>Pixels modified</th>
<th>PSNR of watermarked image</th>
<th>MSE on modified pixels</th>
<th>Number of modified pixels</th>
<th>( \rho ) chosen</th>
</tr>
</thead>
<tbody>
<tr>
<td>30x30=900</td>
<td>2.25%</td>
<td>67.45</td>
<td>0.502</td>
<td>468/931</td>
<td>35</td>
</tr>
<tr>
<td>40x40=1600</td>
<td>4.00%</td>
<td>64.90</td>
<td>0.522</td>
<td>840/1608</td>
<td>29</td>
</tr>
<tr>
<td>50x50=2500</td>
<td>6.25%</td>
<td>63.00</td>
<td>0.517</td>
<td>1302/2517</td>
<td>24</td>
</tr>
<tr>
<td>60x60=3600</td>
<td>9.00%</td>
<td>60.29</td>
<td>0.498</td>
<td>1907/3824</td>
<td>19</td>
</tr>
<tr>
<td>70x70=4900</td>
<td>12.50%</td>
<td>61.35</td>
<td>0.493</td>
<td>2429/4920</td>
<td>16</td>
</tr>
<tr>
<td>80x80=6400</td>
<td>16.00%</td>
<td>60.29</td>
<td>0.493</td>
<td>3414/6925</td>
<td>12</td>
</tr>
<tr>
<td>90x90=1800</td>
<td>20.25%</td>
<td>58.82</td>
<td>0.499</td>
<td>4129/8273</td>
<td>10</td>
</tr>
<tr>
<td>100x100=10000</td>
<td>25.00%</td>
<td>57.14</td>
<td>0.500</td>
<td>5026/10046</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 6.1: Size of logo embedded vs. imperceptibility

6.5 Threshold Selection and Reconstruction of Images

For the experiments above, appropriate \( \rho \)s that produce the critical set of size equal to the logo size, had been chosen by trial and error method. Although this method of choosing \( \rho \) is a time consuming effort which involves a range of repeated experiments,
a correct \( r \) that produces the correct size critical set is vital in order to spread the watermark (logo) uniformly on the image. Inappropriate \( r \) selection would lead to partial embedding of the logo or partial embedding on the image. Best strategy for selection of \( r \) would be to include a binary search algorithm which starts with \( r = 0 \) that includes all points of an image and a \( r = 256 \) that includes none of the points. By imposing a limit on the number of iteration, a \( r \) that is close to the one that produces the correct size critical set can be obtained without incurring much cost.

Since the critical sets produced from the original image and the watermarked image may differ due to distortion caused by watermark embedding, exact reconstruction of the logo would not be possible unless the original critical set is known. In the proposed scheme, the reconstruction of the logos are from recovered critical set on the watermarked image and the originals are not used. Since these critical sets are not the same as the ones produced on the original image, exact recovery of the logo may be impossible.

### 6.6 Synchronization Loss

In this scheme, each and every critical point of the watermarked image is used in the recovery of logo or Edit Distance \( ED(n,m) \) computation even if some points are not in the critical set of the original image. The existence of some extra points or the miss of some original points in the critical set of the watermarked image leads to \emph{synchronization loss} during logo recovery.

For a better explanation, let us give a precise definition of synchronization loss.

"While raster scanning the critical points of a watermarked image, encountering a point which is not in the original critical set or missing a point which is in the original critical set is defined as synchronization loss. That is, if there exist an \( i \) such that the pixel coordinate of \( p_i \) is not equal to the pixel coordinate of \( r_i \), where \( S_{cp} = \{p_1, p_2, \ldots, p_n\} \) are the critical pixels of the original image and \( S_{rp} = \{r_1, r_2, \ldots, r_n\} \) are the critical pixels of the watermarked image"

### 6.7 Advantages

The proposed fragile watermarking scheme tries to avoid the attacks proposed for Yeung-Mintzer scheme. To eliminate this attacks, we embed the watermark on the critical points, which are uniquely determined on an image, in such a way that the
critical points are not disturbed during the verification process unless the image has been tampered with. Even if the same logo and table look up are used on different images, choosing the critical points unique to distinct image avoids the this attack by not revealing the exact locations of embedded marks or pixel positions. The method of embedding marks unique to each image will also avoid collage attack, which involves combination of marked pixels in determining or guessing the logo and look up table. That is, it gives some protection against collage attack because combining portions of different images wouldn't give the same sequence of critical points as the original one.

### 6.8 Drawback and Possible Improvements

This scheme will not reveal the portions of the image that have been tampered with on the logo extracted. A small change on the watermark image that disturbs some critical points will damage the logo starting from those pixel points that represent the damage. Although it might be possible to detect the first occurrence of the manipulation, localizing all the manipulations will be impossible due to synchronization loss. A possible remedy is to divide the image into blocks and embed the logo into each block. Embedding logos into multiple blocks will help in localizing manipulations.

Even on unmanipulated images, we have seen synchronization loss during watermark recovery due to change in critical points caused by embedding watermark. The change could either add some extra critical points or destroy some existing critical points to the critical set determined on the original image. A possible remedy is to construct a look up table in such a way that the critical pixels are not modified very much during embedding. This could be accomplished by employing look up tables with alternating 0s and 1s which allow the maximum change to be atmost 1 so the image will not be very different from the original. Constructing such a look up table will reduce the damage done to the critical set and hence to the logo recovered, while reducing the overall security level of the scheme. Because of the conflicting nature of small change and lower security, there needs to be a compromise between small change (imperceptibility) and security.

Another drawback is that image regions with higher spatial frequency will have more critical points than those of uniform regions. This may cause the problem that particular parts of the image data may not be protected by the proposed watermarking scheme. A possible remedy is to choose critical points at several threshold levels by selecting $p$s at different ranges of $(0,255)$. This method of selecting critical points will include more points from uniform regions and protect those regions as well.
6.9 Possible Extensions: Multiple Watermarks

6.9.1 Threshold Based

Since the peaks and valleys of the critical points can be obtained at various threshold (amplitude) levels, it should be possible to embed multiple watermarks. For example, if a set of peak and valley points are captured in the range of 0-20 (0 ≤ ρ ≤ 20) forming a critical set one, and another set of peak and valley points are captured in the range of 50-70 (50 ≤ ρ ≤ 70) forming a critical set two, it should be possible to embed two logos, as long as there are enough points in those critical sets. As long as peak points are increased and valley points are decreased without disturbing the critical set at different threshold ranges, these watermarks should survive during recovery.

Figure 6.2 shows the two logos that are embedded in Peppers image, choosing ρ at the range of 25 to 35 for critical set one and at the range of 35 to 255 for critical set two and the same logos extracted from the watermarked image.

Figure 6.2: (a) and (b) are embedded logos and (c) and (d) are extracted logos

6.9.2 Region Based

Embedding watermarks (logos) on multiple spatial regions of an image is called *region based watermarking*. Since the peaks and valleys of the critical points can be obtained at different spatial regions, it should be possible to embed multiple watermarks on those regions. For example, if a set of critical points are captured in a spatial region $R_1$ forming a critical set one and another set of critical points are captured in a spatial region $R_2$ forming a critical set two, it should be possible to embed two logos by choosing appropriate $\rho_1$ and $\rho_2$, respectively. Using k-means on feature vectors of pixel points for segmenting watermarked images might end up in wrong region recovery, and hence lead to synchronization loss. Voronoi diagram as used in [72] is a favorable one for segmenting watermarked images, because the regions would not change due to watermark embedding. This is because the centroids on pixel positions, rather than the pixel values, are used as keys.

Figure 6.3 shows the three logos that are embedded in Peppers image after the image has been segmented into three regions by k-means algorithm using the centroids
(23,88), (115,39) and (93,162) trained on image pixel locations. The $\rho$s are set to the values of 20, 21, and 26 to obtain critical sets of size equivalent to the size of the logo. The segmented regions and the three logos that are recovered have also been shown.

![Figure 6.3: (a) Three Voronoi regions chosen (b) three embedded logos and (c) three extracted logos](image)

Even in this instance, the choice of $\rho$s affect the results of these experiment. If $\rho$s are set to the values less than 20, 21, and 26, embedded logos would not cover the entire regions, and if $\rho$s are set to the values greater than 20, 21, and 26, only partial embedding of logo will happen.

### 6.9.3 A Comparison between Threshold and Region Based Methods

The experiments on multiple watermark embedding show region based ones perform better than amplitude based ones. When Voronoi diagrams or the regions segmented by k-means on pixel positions are used, the possibility of watermark embedded in one region disturbing the watermark in another region is less. This is in contrast to our threshold based method, where the chances of critical point movement from one threshold band to another is higher if the bands are not adjusted properly after embedding (ie. before extraction). This is because embedding changes the values of peak and valley points.

### 6.10 Security Analysis

To secure our watermarking system against possible attacks, we have to choose some parameters of the system as keys. The possible parameters that could be chosen as keys are $\rho$, compression levels, low-pass filter, binary look up table, and centroids. Let
us describe how good these parameters are for securing our system. To avoid confusion a separate section is introduced to illustrate the size and range of key space.

6.10.1 $\rho$ as a Key

Though keeping $\rho$ as a key is a possibility, it has some shortcomings. We believe the shortcomings are as follows.

1. If the algorithm to find the critical set is known or public except for the key $\rho$, it would not be difficult to guess $\rho$ from an image critical set pair. If $\rho$ is not guessed correctly and if a small percentage of critical points is missed, recovery of exact logo will be impossible.

2. We recommended earlier that the $\rho$ which had been chosen must produce a critical set with the size equivalent to the size of the logo in order for the logo to be embedded uniformly on the image. If we violate this condition and choose a logo of any other size, it will lead to partial embedding of the logo or partial embedding on the image. In case the logo is smaller, it could be repeated for embedding. Suppose the logo is bigger, it could be subsampled.

6.10.2 Compression Levels as Keys

Keeping compression levels as keys and then choosing the $\rho$ that gives the correct size critical set will not only avoid attacks, but also protect $\rho$ from being guessed. This is because choosing different compression levels does not always lead to the same critical set. We have noticed this from some of our experiments (Table 6.2). This is also not a very good key selection because the overlap between two critical sets derived from two sets of compression levels is almost close to 100%.

| Comp. levels       | $|CS|$ | Common $|CS|$ | $\%$ | $\rho$ |
|--------------------|------|-------------|------|-------|
| 90, 80 and 70      | 1303 | 85%         | 27   |
| 80, 70 and 50      | 1139 | 98%         | 27   |
| 90, 80, 70 and 50  |      | 27          |      |
| 80, 70 and 50      | 1140 | 75%         | 27   |
| 70, 50 and 25      | 880  | 97%         | 27   |
| 80, 70, 50 and 25  | 850  | 27          |      |

Table 6.2: The number of critical points common to different combination of compression levels
From this description, we could conclude that for images that produce non-overlapping critical sets on different sets of compression levels, compression levels could be used as keys.

### 6.10.3 Low-pass Filter as a Key

If an image has been watermarked using a filter that produced a critical set $C$, using another filter for watermark recovery would not produce the same critical set as $C$ even if the size of $C$ is kept constant (by adjusting $\rho$). Hence the recovered watermark would not be the same as the one that was embedded.

For a fixed size critical set, two distinct filters produce more than 36% of non overlapping points. That is, by knowing a critical set produced using a filter, it is extremely hard to guess another critical set produced using a different filter. This is also evident from some experiments we have performed (Table 6.3). Since there are numerous filters we can choose by selecting different basis functions, keeping low pass filter (ie. filter and its properties) as a key is an attractive idea for avoiding attacks.

<table>
<thead>
<tr>
<th>Lenna</th>
<th>DB12 (922)</th>
<th>DB20 (811)</th>
<th>SP (837)</th>
<th>DCT (888)</th>
<th>Canny (939)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB6 (915)</td>
<td>592 (64%)</td>
<td>344</td>
<td>178</td>
<td>223</td>
<td>10</td>
</tr>
<tr>
<td>DB12 (922)</td>
<td>541</td>
<td>275</td>
<td>382</td>
<td>257</td>
<td>10</td>
</tr>
<tr>
<td>DB20 (811)</td>
<td></td>
<td>171</td>
<td>109</td>
<td>14</td>
<td>5</td>
</tr>
<tr>
<td>SP (837)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DCT (888)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6.3: The number of critical points common to different methods of lowpass filterings

### 6.10.4 Binary Look Up Table as a Key

It has already been known from [69] that if the critical points and the sequence of embedding on those points are known for two or more images that are embedded the same logo, the binary look up table can easily be guessed. This attack has been explained in Section 6.2.6. Therefore, major security of our scheme must come from hiding the critical points rather than hiding the binary lookup table. As discussed in the last section hiding the critical points could be achieved by keeping the low-pass filter as a key.

We also recommended before that alternating zeros and ones are needed in the
binary look up table to reduce the synchronization loss during recovery. If the synchronization loss is to be minimized, the binary look up table must become close to 1010... or 0101.... This in turn will reduce the possibility of choosing a random binary sequence in the look up table, and thereby reduce the security of the scheme.

We did experiments to verify how much watermarked images and extracted logos would change from the original ones for different sequences of 0’s and 1’s in the look up table. As expected highest alternating look up tables (ie. 1010... or 0101...) produced the best PSNR values for both watermarked images and logos extracted. This was because highest alternating look up tables produced the least changes in pixel values during embedding. (Table 6.4 and Figure 6.4)

| LUT      | PSNR (Image) | PSNR (Logo) | |CSB| |CSA| |Common| |ρ     |
|----------|--------------|-------------|-------|-------|-------|----------|-------|
| Random   | 59.00        | 12.226      | 931   | 941   | 930   | 35       |
| 010101...| 67.54        | 18.151      | 931   | 932   | 930   | 35       |
| 101010...| 67.55        | 18.399      | 931   | 932   | 929   | 35       |
| 001100...| 63.37        | 14.073      | 931   | 930   | 926   | 35       |
| 110011...| 63.87        | 14.296      | 931   | 932   | 930   | 35       |
| 000111...| 56.61        | 11.798      | 931   | 943   | 930   | 35       |
| 111000...| 57.05        | 14.791      | 931   | 930   | 926   | 35       |

Table 6.4: Look up tables (LUT), PSNR values of watermarked image and extracted logo, size of critical set before (|CSB|) and after embedding (|CSA|) and the number of common points, and the ρ used

Figure 6.4: (a) Original logo and the logo extracted from the watermarked image when (b) random LUT (c) LUT with 0101.. (d) LUT with 1010.. (e) LUT with 0011.. (f) LUT with 1100.. (g) LUT with 000111 and (h) LUT with 111000 are used
6.10.5 Centroids as Keys

In our region based method, there can be several regions chosen depending on the number of centroids used. The locations of those centroids can be numerous. The number of centroids determines the number of logos that could be embedded. The locations of those centroids determine the regions in which the logos are embedded. Since the regions chosen are arbitrary, the centroids of those regions can be used as keys.

6.11 Size of Key Space

6.11.1 \( \rho \) as a Key

Since the maximum value of pixel value difference between the original image and the lowpass image is 255, \( \rho \) can take values ranging from 0 to 255. Keeping \( \rho \) as 0 will include all pixel points as critical points and keeping \( \rho \) as 255 will include none of the pixel points as critical points. Therefore, key space of \( \rho \) is upper bounded by the value of 255.

6.11.2 Compression Levels as Keys

JPEG compression quality level range from 0 to 100. Choosing quality levels below certain value \( l \) may not produce good quality images and so lead to great proportion of insignificant critical points. Suppose \( l \) has been set to an appropriate quality level (close to 100), there are \( 100 - l \) possible values to choose from. Since a quality level chosen to produce a set of points should not be repeated for producing another set of points, each level chosen must be distinct in the range of \( l \) to 100. If \( p \) (less than or equal to \( 100-l+1 \)) levels are chosen, the size of key space will be \( (100-l+1)(100-l)\ldots(100-l-p+2) \) which is equal to \( \frac{(100-l+1)!}{(100-l-p+1)!} \).

6.11.3 Low-pass Filter as a Key

In the case of DCT, there are 64 DCT coefficients available for each 8x8 block. In low pass filtering using DCT, if \( m \) high frequency coefficients are made 0 uniformly among all blocks, there are \( m \) possible keys. If the number of high frequency coefficients that are made 0 is varied among blocks and kept within the range of 1 to \( m \), assuming \( n \) blocks are there, the size of the key space will be less than or equal to \( m^n \).

In the case of wavelet, the number of possible low-pass wavelet coefficients are
numerous. This is because the choice of basis functions are numerous. If this scheme is implemented as a part of JPEG 2000, the choice of wavelet coefficients will be restricted to the choice of wavelet coefficients used for JPEG.

6.11.4 Binary Look Up Table as a Key

There are altogether $2^{256}$ possible look up tables. Out of these tables, The best ones are 0101.... and 1010.... If the maximum number of consecutive 0’s and 1’s are kept within 3, there are several possible tables to choose from.

6.12 A Comparison Between Bhattacharjee’s Scheme and Ours

Since Bhattacharjee’s scheme which is found in [72] is the only region or feature point based watermarking scheme, known to the author, which some what resembles our multiple watermaring scheme, we give a comparison between our scheme and theirs.

1. Their scheme uses Mexican-hat wavelet for feature point detection, whereas ours uses Daubachie/Biorthogonal wavelet or DCT for feature extraction.

2. Their scheme does not take compression levels into consideration, whereas ours takes a range of compression levels into consideration before the feature points are determined.

3. In their scheme, Voronoi regions are formed choosing the feature points determined as centroids. In our scheme, Voronoi regions are determined based on centroids trained on arbitrary initial points.

4. Spread spectrum watermarking has been recommended for marking each region in their scheme. We use Yeung-Mintzer scheme for embedding marks on critical points.

5. They do not have sufficient explanation for choosing the feature points as centroids of Voronoi regions. Since our choice of critical points (feature points) are the high frequency points of an image, embedding marks on those points will go unnoticed.

6. Their main reason for watermarking is copyright protection, thus robustness plays a vital role than fragility. Our main reason for watermarking is authentication, thus fragility plays a vital role than robustness.
7. Their scheme has not been made key dependent for some reason. Our scheme is made key dependent by hiding the critical points generated as well as the regions chosen for embedding.

8. In their case, multiple spread spectrum can be embedded by choosing each one for each region, whereas in ours, multiple logos can be embedded by choosing each logo for each region.

6.13 Performance Evaluation

We determine if an image has been manipulated or not by comparing the embedded logo sequence with the extracted logo sequence. This could be done either by visually comparing the embedded logo with the extracted logo or by measuring the Edit Distance between the embedded logo sequence and the extracted logo sequence. The correlation between the visual judgement and the distance measure is uncertain and needs further experiments to validate this uncertainty. The recovery of the logo may be exact or somewhat close to the one which was embedded. If it is an exact recovery where $ED(n,m) = \tau$, we say the image has not been subject to manipulations, and if it is somewhat close, where $ED(n,m) < \tau$, we say the image has been subject to tolerable manipulation.

To evaluate the performance of our scheme, after embedding a logo, we performed the following attacks on the watermarked image: JPEG compression at quality level of 90, blurring, embossing, horizontal flipping, oil-painting, rotation to one degree, vertical flipping, and sharpening. We embedded a logo of size $30 \times 30$ (900 bits), extracted the same logo, and computed the Edit Distance between the embedded and extracted logos. The Edit distance (ED), the size of the logo embedded (SOL), and the size of the logo extracted (SEL) are illustrated in Table 6.5.

Out of these attacks, we were able to recover the embedded logo in a distorted form from the watermarked images for the following attacks: JPEG compression at quality level of 90, horizontal flipping, oil-painting, and vertical flipping. This was because the size of the critical set remained almost the same before and after embedding or increased from its original size. For blurring, embossing, and rotation to one degree attacks, the size of the recovered critical set was considerably reduced from the original ones. For sharpening attack, the size of the recovered critical set was considerably increased from the original one. Hence we could not recover the embedded logo even in distorted form. For those attacks for which we recovered the logo are shown in Figure
6.13. Performance Evaluation

<table>
<thead>
<tr>
<th>LUT</th>
<th>ED</th>
<th>SOS</th>
<th>SEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>41</td>
<td>900</td>
<td>943</td>
</tr>
<tr>
<td>JPEG-90</td>
<td>353</td>
<td>900</td>
<td>994</td>
</tr>
<tr>
<td>Blurring</td>
<td>808</td>
<td>900</td>
<td>94</td>
</tr>
<tr>
<td>Emboss</td>
<td>788</td>
<td>900</td>
<td>714</td>
</tr>
<tr>
<td>HorFlip</td>
<td>164</td>
<td>900</td>
<td>943</td>
</tr>
<tr>
<td>OilPaint</td>
<td>1037</td>
<td>900</td>
<td>1939</td>
</tr>
<tr>
<td>Rotate</td>
<td>.410</td>
<td>900</td>
<td>662</td>
</tr>
<tr>
<td>VerFlip</td>
<td>522</td>
<td>900</td>
<td>943</td>
</tr>
<tr>
<td>Sharpen</td>
<td>1808</td>
<td>900</td>
<td>2710</td>
</tr>
<tr>
<td>Modify</td>
<td>160</td>
<td>900</td>
<td>1062</td>
</tr>
</tbody>
</table>

Table 6.5: Edit Distances between embedded and extracted logos, size of logo embedded, and size of logo extracted.

6.5. The recoverable logos in distorted form and the irrecoverable logos for the attacks mentioned above show the performance of our fragile watermarking scheme.

![Recovered Logos](image1.png)

Figure 6.5: Logo recovered from (a) JPEG compression at quality level 90 (b) horizontal flipping (c) oilpainting and (d) vertical flipping

**6.13.1 Edit Distance as a Measure of Distortion**

Although Figure 6.5 shows the visual impact of some of the attacks on the embedded logo or watermark, it does not quantify the amount of distortion caused to the logo or image. Since the size of the critical set computed from the attacked image varies considerably from the size of the critical set computed on the original image, it wouldn’t be possible to recover a logo sequence which is of the same size as the embedded logo. This also means we cannot use PSNR (Peak Signal to Noise Ratio) or HD (Hamming Distance) as a measure of image (logo) distortion for this scheme. Since PSNR for gray scale or color images and HD for binary images measure the distortion well between two signals only when the signals are of the same size and since PSNR is not at all a valid measure for measuring the quality of binary images (logos), Edit Distance (ED) [80] as opposed to PSNR and HD has been chosen to quantify the amount of distortion caused to a logo.
6.13.2 Statistical Hypothesis Testing

Our objective now is to evaluate the performance of our fragile watermarking scheme with respect to the attacks mentioned before using statistical hypothesis of testing. Here we use look up tables with alternating 0’s and 1’s in such a way that the highest modification of the critical pixels is kept within the value of 3. This is not the case with randomly generated 0’s and 1’s. To determine a confidence level on the decision we use the following steps.

Assume the Edit Distances between the logo extracted from the unmanipulated image and the original logo that has been embedded belonging to admissible set, follow normal distribution with mean \( \bar{\sigma} \) and standard deviation \( \sigma_x \), and the Edit Distances between the logo extracted from the manipulated image and the original logo that has been embedded belonging to inadmissible set, follow normal distribution with mean \( \bar{\sigma} \) and standard deviation \( \sigma_o \). We shall now define a null hypothesis and an alternative hypothesis as follows.

- **Hypothesis** \( H_0 \): Image under consideration is accepted as original if the Edit Distance \( ED(n,m) \) computed between \( O_n \) and \( R_m \) falls within the range of \([a, b]\).

- **Hypothesis** \( H_1 \): Image under consideration is rejected as manipulated if the Edit Distance computed between \( O_n \) and \( R_m \) falls out of the range of \([a, b]\).

Rejecting \( H_0 \) when it should be accepted (i.e. \( \Pr(\text{Type 1 error}) = a \)) is the level of significance of the test of \( H_0 \). The probability of accepting \( H_o \) when actually an image is manipulated is given by area \( \beta \).

Assuming the level of significance is \( \alpha = 0.1 \) and using the following statistics:

1. Compute \( \bar{\sigma} \) and \( \sigma_x \) on Edit Distances \( ED(n,m) \) calculated on logos extracted from unmanipulated images using more than 30 different LUTs.

2. Compute \( \bar{\sigma} \) and \( \sigma_o \) on Edit Distances \( ED(n,m) \) calculated on logos extracted from manipulated images using more than 30 different LUTs.

We could compute the acceptance region for \( H_0 \), which will result in the range of \([a, b]\). This indicates the rejection region for \( H_0 \) would be the region other than \([a, b]\). We could also compute the probability of accepting \( H_0 \) when actually an image was manipulated (i.e., \( \beta \)).

The range and \( \beta \) showed in Table 6.6 are only approximates because we have not considered enough sample data.
<table>
<thead>
<tr>
<th>LUT</th>
<th>ran</th>
<th>0101</th>
<th>0110</th>
<th>0011</th>
<th>1100</th>
<th>000111</th>
<th>111000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>41</td>
<td>32</td>
<td>34</td>
<td>38</td>
<td>33</td>
<td>44</td>
<td>38</td>
</tr>
<tr>
<td>JPEG-90</td>
<td>353</td>
<td>359</td>
<td>372</td>
<td>355</td>
<td>379</td>
<td>387</td>
<td>364</td>
</tr>
<tr>
<td>Blurring</td>
<td>808</td>
<td>813</td>
<td>812</td>
<td>810</td>
<td>812</td>
<td>805</td>
<td>805</td>
</tr>
<tr>
<td>Emboss</td>
<td>788</td>
<td>790</td>
<td>789</td>
<td>787</td>
<td>788</td>
<td>786</td>
<td>783</td>
</tr>
<tr>
<td>HorFlip</td>
<td>164</td>
<td>157</td>
<td>160</td>
<td>161</td>
<td>160</td>
<td>165</td>
<td>159</td>
</tr>
<tr>
<td>OilPaint</td>
<td>1037</td>
<td>1004</td>
<td>995</td>
<td>1037</td>
<td>474</td>
<td>1067</td>
<td>1060</td>
</tr>
<tr>
<td>Rotate</td>
<td>410</td>
<td>412</td>
<td>406</td>
<td>429</td>
<td>408</td>
<td>423</td>
<td>379</td>
</tr>
<tr>
<td>VerFlip</td>
<td>522</td>
<td>522</td>
<td>522</td>
<td>524</td>
<td>522</td>
<td>522</td>
<td>523</td>
</tr>
<tr>
<td>Sharpen</td>
<td>1808</td>
<td>1819</td>
<td>1815</td>
<td>1813</td>
<td>1818</td>
<td>1814</td>
<td>1828</td>
</tr>
<tr>
<td>Modify</td>
<td>160</td>
<td>183</td>
<td>198</td>
<td>154</td>
<td>155</td>
<td>148</td>
<td>155</td>
</tr>
</tbody>
</table>

$\sigma_c \quad 4.46$  
$\bar{\sigma} \quad 664.57$  
$\sigma_o \quad 494.75$  
$\text{Range}[a,b] \quad [0-42.23]$  
$\beta \quad 0.1038$

Table 6.6: Edit Distances between embedded and extracted logos

### 6.13.3 Real World Attack

Here we illustrate an example of a real world attack where a number plate of a car has been replaced with another (Figure 6.6). Although this is a very small modification, PSNR value and Edit Distance show the difference very well. The number of pixels modified are around 200 out of 53200 pixels (0.406%). The PSNR value between the watermarked image and the modified watermarked image is 36.32. The Edit Distance computed on modified or attacked watermarked image differs from the Edit Distance computed on watermarked image by the value of 11.
An experiment to evaluate the correlation between the extracted string sizes or Edit distances, and the size of the logo embedded is performed and the table below shows the results.

For this experiment car image has been embedded different size logos (30 × 30, 40 × 40, 50 × 50, 60 × 60) after choosing appropriate size critical sets with \( \rho = 126, 113, 101, \) and 91, respectively. The same modification is done on all those embedded images. The sizes of extracted string before and after modification are noted. The Edit distances between the embedded and extracted logo before and after modifications are also noted. Experimental results are illustrated in Table 6.7.

<table>
<thead>
<tr>
<th>( \rho )</th>
<th>Logo Size</th>
<th>Extracted String Size (Unmodified)</th>
<th>Extracted String Size (Modified)</th>
<th>Edit Distance (Unmodified)</th>
<th>Edit Distance (Modified)</th>
<th>PSNR</th>
<th>MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>126</td>
<td>30x30=900</td>
<td>951</td>
<td>963 (12)</td>
<td>126</td>
<td>137 (11)</td>
<td>36.32</td>
<td>14.78</td>
</tr>
<tr>
<td>113</td>
<td>40x40=1600</td>
<td>1639</td>
<td>1657 (18)</td>
<td>176</td>
<td>193 (17)</td>
<td>35.56</td>
<td>18.05</td>
</tr>
<tr>
<td>101</td>
<td>50x50=2500</td>
<td>2526</td>
<td>2546 (20)</td>
<td>202</td>
<td>221 (19)</td>
<td>37.25</td>
<td>12.22</td>
</tr>
<tr>
<td>91</td>
<td>60x60=3600</td>
<td>3702</td>
<td>3732 (30)</td>
<td>1235</td>
<td></td>
<td>35.50</td>
<td>18.32</td>
</tr>
<tr>
<td>70x70=4900</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6.7: PSNR, MSE, embedded and extracted string sizes, and the Edit Distances are shown

The difference between the Edit Distance of unmodified and the Edit Distance of the modified image has a positive correlation with the size of the logo embedded. This also means the larger the logo embedded the better the detection performance.

6.13.4 Type 1 and Type 11 Errors

In the next two sections, we construct some examples to show false negative and false positive results. For this, the following assumptions are made.

- If the PSNR computed on a watermarked image is less than 40, the image will be considered as manipulated.

- If the Edit Distance is less than 300, accept the image as unmanipulated, else reject it as manipulated.

The decision on whether an image has been manipulated or not is based on the PSNR value of the image and the decision on whether to accept or reject a received
image as unmanipulated or not is made on the Edit Distance computed on the logo extracted. Table 6.7 shows the thresholds we use to make these decisions.

<table>
<thead>
<tr>
<th></th>
<th>PSNR&lt;40 (Manipulated)</th>
<th>PSNR&gt;40 (Unmanipulated)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ED&lt;300 (Accept)</td>
<td>Type 11 (FN)</td>
<td>-</td>
</tr>
<tr>
<td>ED&gt;300 (Reject)</td>
<td>-</td>
<td>Type 1 (FP)</td>
</tr>
</tbody>
</table>

Table 6.8: Type 1 and Type 11 Errors

**False Negatives (FN)**

There are situations in which an image has been manipulated, and the decision is made to accept the image as unmanipulated. Horizontal flip, copy and paste 1, removal, horizontal line drawing, and copy and paste 2 are the best examples illustrating this situation. (Tables 6.8 and 6.9) For these manipulations PSNR remains at 11.51, 27.23, 18.63, 25.76, and 21.94, respectively, indicating image has been manipulated, while the Edit Distance of 157, 129, 135, 261, and 227 indicating the acceptance decision.

Although these manipulations do not show the visual impact on the images, the experimental results illustrate false negative results are possible if the image has been manipulated without disturbing the critical points.
False Positive (FP)

There are situations in which an image has been partially manipulated or unmanipulated (i.e. one can recognize the content of the image), and the decision is made to reject the image as manipulated. JPEG compression at quality levels of 95 and 98 are the best examples illustrating this situation. For these manipulations, PSNR remains at 44.27 and 49.75, respectively, indicating image has not been manipulated much, while the Edit Distances of 355 and 321, indicating the reject decision. In these circumstances, appropriate threshold on Edit Distances has to be set to reduce the error. (Tables 6.8 and 6.9) Other manipulations in this table shows the false negative results. The experimental results show, by setting the threshold on Edit Distance, various levels of authenticity could be achieved.

<table>
<thead>
<tr>
<th>Image</th>
<th>Pixels modified</th>
<th>Edit Distance</th>
<th>PSNR</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hor-flip</td>
<td>28x32=896</td>
<td>157</td>
<td>11.51</td>
<td>FN</td>
</tr>
<tr>
<td>Copy-paste-1</td>
<td>33x37=1221</td>
<td>129</td>
<td>27.23</td>
<td>FN</td>
</tr>
<tr>
<td>Remove</td>
<td>2x200=400</td>
<td>135</td>
<td>18.63</td>
<td>FN</td>
</tr>
<tr>
<td>Hor-line</td>
<td>58x54=3132</td>
<td>261</td>
<td>25.76</td>
<td>FN</td>
</tr>
<tr>
<td>Copy-paste-2</td>
<td>200x2=400</td>
<td>227</td>
<td>21.94</td>
<td>FN</td>
</tr>
<tr>
<td>Ver-line</td>
<td>JPEG-100</td>
<td>614</td>
<td>26.23</td>
<td></td>
</tr>
<tr>
<td>JPEG-98</td>
<td></td>
<td>141</td>
<td>58.12</td>
<td>FN</td>
</tr>
<tr>
<td>JPEG-95</td>
<td></td>
<td>321</td>
<td>49.75</td>
<td>FP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>355</td>
<td>44.27</td>
<td>FP</td>
</tr>
</tbody>
</table>

Table 6.9: Edit Distances and PSNR between images
6.14 Conclusion

In this chapter, we have proposed a new fragile watermarking scheme which embeds the marks on critical points which are obtained by detecting the high frequency points of an image. These critical points are intact even after embedding the marks unless the image has been tampered with. This scheme prevents the two main attacks proposed for Yeung-Mintzer’s scheme. The security level of the scheme and the possible extension to multiple watermarking scheme have also been investigated. Edit Distance as opposed to PSNR or HD has been recommended as a performance measure of the scheme.
<table>
<thead>
<tr>
<th>Method</th>
<th>Image</th>
<th>CP on Original</th>
<th>CP on Watermark</th>
<th>Common CP</th>
<th>PSNR</th>
<th>( \rho )</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB3</td>
<td>Lenna</td>
<td>916</td>
<td>914</td>
<td>911</td>
<td>61.24</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Peppers</td>
<td>1076</td>
<td>1069</td>
<td>1066</td>
<td>60.75</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Pills</td>
<td>899</td>
<td>897</td>
<td>897</td>
<td>34.50</td>
<td>70</td>
</tr>
<tr>
<td></td>
<td>Paper</td>
<td>792</td>
<td>788</td>
<td>788</td>
<td>41.88</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Corrosion</td>
<td>579</td>
<td>574</td>
<td>573</td>
<td>33.57</td>
<td>50</td>
</tr>
<tr>
<td>DB6</td>
<td>Lenna</td>
<td>923</td>
<td>915</td>
<td>915</td>
<td>60.70</td>
<td>70</td>
</tr>
<tr>
<td></td>
<td>Peppers</td>
<td>1136</td>
<td>1130</td>
<td>1130</td>
<td>60.22</td>
<td>70</td>
</tr>
<tr>
<td></td>
<td>Pills</td>
<td>864</td>
<td>863</td>
<td>863</td>
<td>34.50</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Paper</td>
<td>757</td>
<td>760</td>
<td>754</td>
<td>41.02</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td>Corrosion</td>
<td>536</td>
<td>535</td>
<td>533</td>
<td>32.55</td>
<td>70</td>
</tr>
<tr>
<td>DB12</td>
<td>Lenna</td>
<td>812</td>
<td>807</td>
<td>805</td>
<td>61.17</td>
<td>90</td>
</tr>
<tr>
<td></td>
<td>Peppers</td>
<td>1115</td>
<td>1104</td>
<td>1103</td>
<td>60.12</td>
<td>90</td>
</tr>
<tr>
<td></td>
<td>Pills</td>
<td>898</td>
<td>897</td>
<td>897</td>
<td>34.50</td>
<td>120</td>
</tr>
<tr>
<td></td>
<td>Paper</td>
<td>739</td>
<td>732</td>
<td>731</td>
<td>41.89</td>
<td>93</td>
</tr>
<tr>
<td></td>
<td>Corrosion</td>
<td>584</td>
<td>582</td>
<td>581</td>
<td>32.55</td>
<td>80</td>
</tr>
<tr>
<td>BO</td>
<td>Lenna</td>
<td>838</td>
<td>828</td>
<td>827</td>
<td>60.64</td>
<td>125</td>
</tr>
<tr>
<td></td>
<td>Peppers</td>
<td>1245</td>
<td>1244</td>
<td>1241</td>
<td>60.46</td>
<td>120</td>
</tr>
<tr>
<td></td>
<td>Pills</td>
<td>852</td>
<td>848</td>
<td>848</td>
<td>34.50</td>
<td>145</td>
</tr>
<tr>
<td></td>
<td>Paper</td>
<td>745</td>
<td>740</td>
<td>740</td>
<td>41.88</td>
<td>115</td>
</tr>
<tr>
<td></td>
<td>Corrosion</td>
<td>573</td>
<td>568</td>
<td>568</td>
<td>32.55</td>
<td>95</td>
</tr>
<tr>
<td>DCT</td>
<td>Lenna</td>
<td>889</td>
<td>881</td>
<td>878</td>
<td>61.40</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>Peppers</td>
<td>884</td>
<td>873</td>
<td>872</td>
<td>61.38</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>Pills</td>
<td>884</td>
<td>881</td>
<td>881</td>
<td>34.50</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>Paper</td>
<td>744</td>
<td>741</td>
<td>740</td>
<td>41.88</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>Corrosion</td>
<td>547</td>
<td>541</td>
<td>541</td>
<td>32.55</td>
<td>36</td>
</tr>
</tbody>
</table>

Table 6.10: Critical points common to both original and watermarked images considering compression quality levels of 70, 50, and 25
<table>
<thead>
<tr>
<th>Method</th>
<th>Image</th>
<th>CP on Original</th>
<th>CP on Watermark</th>
<th>Common CP</th>
<th>PSNR</th>
<th>$\rho$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Lenna</td>
<td>1505</td>
<td>1497</td>
<td>1490</td>
<td>59.40</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Peppers</td>
<td>1730</td>
<td>1729</td>
<td>1712</td>
<td>59.00</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Pills</td>
<td>1479</td>
<td>1479</td>
<td>1477</td>
<td>37.69</td>
<td>70</td>
</tr>
<tr>
<td></td>
<td>Paper</td>
<td>1713</td>
<td>1719</td>
<td>1700</td>
<td>43.61</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Corrosion</td>
<td>1324</td>
<td>1322</td>
<td>1321</td>
<td>34.19</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Lenna</td>
<td>1263</td>
<td>1269</td>
<td>1254</td>
<td>59.77</td>
<td>70</td>
</tr>
<tr>
<td></td>
<td>Peppers</td>
<td>1616</td>
<td>1606</td>
<td>1598</td>
<td>59.08</td>
<td>70</td>
</tr>
<tr>
<td></td>
<td>Pills</td>
<td>1278</td>
<td>1282</td>
<td>1275</td>
<td>34.49</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Paper</td>
<td>1307</td>
<td>1301</td>
<td>1288</td>
<td>41.85</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td>Corrosion</td>
<td>965</td>
<td>974</td>
<td>965</td>
<td>34.54</td>
<td>70</td>
</tr>
<tr>
<td></td>
<td>Lenna</td>
<td>1061</td>
<td>1052</td>
<td>1051</td>
<td>59.87</td>
<td>90</td>
</tr>
<tr>
<td></td>
<td>Peppers</td>
<td>1416</td>
<td>1412</td>
<td>1409</td>
<td>59.55</td>
<td>90</td>
</tr>
<tr>
<td></td>
<td>Pills</td>
<td>1233</td>
<td>1233</td>
<td>1231</td>
<td>34.49</td>
<td>120</td>
</tr>
<tr>
<td></td>
<td>Paper</td>
<td>1153</td>
<td>1155</td>
<td>1150</td>
<td>41.86</td>
<td>93</td>
</tr>
<tr>
<td></td>
<td>Corrosion</td>
<td>1004</td>
<td>1008</td>
<td>999</td>
<td>32.54</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>Lenna</td>
<td>1168</td>
<td>1152</td>
<td>1148</td>
<td>59.93</td>
<td>125</td>
</tr>
<tr>
<td></td>
<td>Peppers</td>
<td>1706</td>
<td>1708</td>
<td>1705</td>
<td>58.99</td>
<td>120</td>
</tr>
<tr>
<td></td>
<td>Pills</td>
<td>1243</td>
<td>1244</td>
<td>1238</td>
<td>34.49</td>
<td>145</td>
</tr>
<tr>
<td></td>
<td>Paper</td>
<td>1196</td>
<td>1193</td>
<td>1189</td>
<td>41.84</td>
<td>115</td>
</tr>
<tr>
<td></td>
<td>Corrosion</td>
<td>881</td>
<td>882</td>
<td>877</td>
<td>32.54</td>
<td>95</td>
</tr>
<tr>
<td></td>
<td>Lenna</td>
<td>1974</td>
<td>1987</td>
<td>1967</td>
<td>58.16</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>Peppers</td>
<td>1935</td>
<td>1948</td>
<td>1930</td>
<td>58.15</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>Pills</td>
<td>2107</td>
<td>2116</td>
<td>2103</td>
<td>36.35</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>Paper</td>
<td>1414</td>
<td>1418</td>
<td>1407</td>
<td>43.61</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>Corrosion</td>
<td>1630</td>
<td>1648</td>
<td>1621</td>
<td>34.18</td>
<td>36</td>
</tr>
</tbody>
</table>

Table 6.11: Critical points common to both original and watermarked images without considering compression
Figure 6.6: Original car image, watermarked car image and the modified watermarked car image
Figure 6.7: Manipulations without disturbing the critical points

Figure 6.8: Modified images
Chapter 7

Fragile Watermark Based on Polarity of Pixel Points

7.1 Introduction

The aim of authentication is to protect the information being transmitted. Cryptography provides the means for accomplishing the authenticity of the information being received/sent and guarantees that the information has not been tampered with. Hash function protects the integrity of the information by impacting much on the hash digest for even a single bit change. For images this level of protection is not so important because a change in a single pixel value will virtually have no impact on the impression of the image. Watermarking as opposed to cryptographic tool provides a sufficient amount of protection while localising the tampered areas without having to store additional information. This is accomplished at the expense of slightly modifying the pixel values.

7.2 Previous Schemes

An early proposal for a fragile watermark is found in [67]. In this, Walton proposes to hide key dependent check-sums of the seven most significant bits (MSB) of gray scales along pseudo random walks in the least significant bits (LSB) of pixels forming the walk. The main drawback of this scheme is that if the random walk is guessed it is easy to attack the image.

Wong’s method of fragile watermarking scheme based on cryptographic hash function is found in [66]. In this he proposes a fragile water marking scheme which divides an image into blocks, calculates the hash value from MSB of the pixels and embeds the hash sequence in the LSB of the pixels. The main drawback of this scheme is that it is subject to collagge attack if the block sizes are known.

In Honsinger’s scheme [71] the original image can be recovered if the watermark
image has been found to be authentic.

7.3 Our Proposal

We propose a new fragile watermarking scheme which tries to avoid the attacks proposed for Yeung-Mintzer scheme. Their scheme is subject to two types of attacks. The first one assumes that two or more images have been embedded with a watermark using the same binary logo and look up table [68] [69]. The second one which is also known as the collage attack assumes that portions of two or more images that have been embedded watermarks using the same binary logo and look up table can be put together to construct a new watermark image. To eliminate the weak points we embed the watermark on the center pixel values of image blocks in such a way that the polarity of those pixel points are not disturbed during the verification process unless the image has been tampered with. Even if the same logo and table look up are used on different images, choosing the polarity unique to each image will avoid the first attack to greater extend. This will happen because the polarity of the center pixel of the image blocks which differ from image to image are taken into consideration during embedding. This method of embedding marks unique to each image will also avoid the combination of marked pixels in determining or guessing the logo and look up table, and also the collage attack. The collage attack is somewhat protected because combining portions of different images wouldn’t give the same sequence of polarity as the original one.

The idea of this fragile watermarking scheme come from Yeung-Mintzer’s scheme and our critical set based fragile watermarking scheme. In our proposed watermarking scheme [9] critical points are derived from taking the difference between the original image and the low pass image and keeping the co-ordinates which have the differences above a certain threshold as critical points.

In this scheme, we determine the polarity of the center pixel of an image block by determining if the difference between the center pixel of the image block and the mean of the image block pixels is greater than zero or not. If the difference is greater than zero, we say it is a peak point, having positive polarity, else we say it is a valley point, having negative polarity. If the difference is a positive one increasing the center pixel value of the image block wouldn’t change the polarity of the center pixel. If the difference is a negative one decreasing the center pixel value of the image block wouldn’t change the polarity of the center pixel. The idea of polarity of image pixel come from [81], where the polarity of VQ indices are XORed with watermarked image to obtain the key. The properties mentioned above will allow the polarity of center pixels of
image blocks to be in tact during watermark recovery. We construct an algorithm preserving the above properties.

### 7.3.1 Watermark Embedding

1. Given an image $I$, divide it into blocks of size $a \times b$.

2. Let $B_1, B_2, ..., B_n$ be the blocks of size $a \times b$ ordered according to raster scan of the image $I$.

3. Let $C_1, C_2, ..., C_n$ be the center pixels of the image blocks $B_1, B_2, ..., B_n$ and let $\overline{x}_1, \overline{x}_2, ..., \overline{x}_n$ be the mean of image blocks $B_1, B_2, ..., B_n$.

4. The polarity of the center pixels $C_1, C_2, ..., C_n$ are determined by finding the difference between $C_i$ and $\overline{x}_i$ for each block $B_i$. If $C_i - \overline{x}_i > 0$, then $C_i$ is said to have positive polarity ('+') , else $C_i$ is said to have negative polarity ('-').

5. Let $P_1, P_2, ..., P_n$ be the polarity of the center pixels $C_1, C_2, ..., C_n$ of the blocks $B_1, B_2, ..., B_n$.

6. Let $L = \{l_1, l_2, ..., l_n\}$ be the raster scanned binary map of the logo.

7. Let $f_{pos}: \{0, 1, .., 255\} \rightarrow \{0, 1\}$ defines the look up table of positive polar center pixels for a particular key.

8. Let $f_{neg}: \{0, 1, .., 255\} \rightarrow \{0, 1\}$ defines the look up table of negative polar center pixels for a particular key.

9. For each center pixel from $C_1$ to $C_n$
   
   - if $f_{pos}(C_i) = l_i$ and $P_i = " + "$ then do not modify $C_i$
   - else if $f_{neg}(C_i) = l_i$ and $P_i = " - "$ then do not modify $C_i$
   - else if $P_i = " + "$ then increase $C_i$ by a value of $\delta$ until $f_{pos}(C_i + \delta) = l_i$
   - else if $P_i = " - "$ then decrease $C_i$ by a value of $\delta$ until $f_{neg}(C_i - \delta) = l_i$

### 7.3.2 Verification Algorithm

1. Upon receiving an image $I^r$, divide it into blocks of size $a \times b$.

2. Let $B_1^r, B_2^r, ..., B_n^r$ be the blocks of size $a \times b$ ordered according to raster scan of the image $I^r$. 
3. Let $C_1^r, C_2^r, \ldots, C_n^r$ be the center pixels of the image blocks $B_1^r, B_2^r, \ldots, B_n^r$ and let $\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n$ be the mean of image blocks $B_1^r, B_2^r, \ldots, B_n^r$.

4. Let $P_1^r, P_2^r, \ldots, P_n^r$ be the polarity of the center pixels $C_1^r, C_2^r, \ldots, C_n^r$ of the blocks $B_1^r, B_2^r, \ldots, B_n$ as defined above.

5. Let $l_1^r, l_2^r, \ldots, l_n^r$ be the logo sequence extracted using the procedure defined in the next step.

6. For each center pixel from $C_1$ to $C_n$
   
   if $P_i = " + "$ then extract $l_i^r$ as $f_{pos}(C_i^r)$ and
   
   if $P_i = " - "$ then extract $l_i^r$ as $f_{neg}(C_i^r)$
   
   For each center pixel from $C_1$ to $C_n$
   
   if $l_i^r = l_i$ then the received image $I^r$ has not been tampered with. if for some blocks $l_i^r = l_i$ do not hold, that portion of the image has been tampered with.

### 7.4 Experimental Results

We performed experiments on Lena, Peppers, Pills, Paper, and Corrosion images to verify if the polarities of the center pixels of the original image blocks remain the same as the polarity of the center pixels of the watermarked image blocks. As expected polarity of the image blocks remained the same. We also looked at the quality of the image after embedding by computing PSNR values. Our experiments revealed that exact recovery of the logo was possible on all five images. PSNR values never became less than 32 on all images. Table 7.1 shows the block sizes chosen for embedding and the PSNR values that measure the image fidelity. For each block size, appropriate size logos were chosen to incorporate entire blocks of the image.

<table>
<thead>
<tr>
<th>Block Size</th>
<th>Lena (200x200)</th>
<th>Peppers (200x200)</th>
<th>Pills (130x200)</th>
<th>Paper (132x200)</th>
<th>Corrosion (137x200)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5x5</td>
<td>65.002</td>
<td>65.090</td>
<td>34.468</td>
<td>41.877</td>
<td>32.514</td>
</tr>
<tr>
<td>10x10</td>
<td>71.318</td>
<td>71.229</td>
<td>34.471</td>
<td>41.893</td>
<td>32.516</td>
</tr>
<tr>
<td>15x15</td>
<td>75.231</td>
<td>74.875</td>
<td>34.472</td>
<td>41.896</td>
<td>32.516</td>
</tr>
<tr>
<td>20x20</td>
<td>77.339</td>
<td>76.908</td>
<td>34.472</td>
<td>41.896</td>
<td>32.517</td>
</tr>
<tr>
<td>25x25</td>
<td>78.13</td>
<td>79.837</td>
<td>34.472</td>
<td>41.897</td>
<td>32.517</td>
</tr>
</tbody>
</table>

Table 7.1: PSNR computed on images choosing five different block sizes
We also embedded the same logo (Figure 7.1(a)) on all five images and extracted them to see the visual quality degradation caused to the logo due to embedding. Figures 7.1(b)-(f) show the extracted logo from Lena, Peppers, Pills, Paper, and Corrosion images.

![Embedded Logos](image)

Figure 7.1: (a) Original Logo (b) Logo on Lena (c) Logo on Peppers (d) Logo on Pills (e) Logo on Paper (f) Logo on Corrosion

### 7.5 Drawbacks and Possible Remedies

This method is subject to similar attacks proposed for Yeung-Mintzer scheme. Some possible protection methods against this attacks are as follows.

1. **Randomize pixel selection within each block:**
   
   This could be done by choosing a random pixel in each block rather than choosing a center pixel.

2. **Randomize block selection in embedding:**
   
   In embedding, instead of choosing the blocks in a raster scan of an image, choose the blocks in a random order.

3. **More than one Look Up Tables (LUTs):**
   
   To enhance protection, choose more than one LUT. For example, LUT one for
center pixels with positive polarity and LUT two for center pixels with negative polarity.

7.6 Advantages of the Scheme

Some advantages of the scheme are as follows:

1. Embedding information in this way on the center pixels of image blocks would not disturb or influence the neighbouring blocks at all. This is because non-overlapping blocks are chosen in the embedding/extraction process.

2. Localization of the manipulations is possible because the blocks used in the embedding/extraction process are uniformly spread over the image.

3. There is no synchronization loss during logo recovery due to embedding information or manipulation of the watermarked image. The size of the embedded logo sequence will always be the same as the size of the recovered logo sequence.

7.7 Possible Improvements: Multiple Watermarks

7.7.1 Variance Based

Since the variance of image blocks can be at various ranges, it should be possible to divide the image into various regions based on block variances. For example, if a set of blocks with variances ranging from 0 to $l_1$ forms a region $R_1$ and a set of blocks with variances ranging from $l_1 + \delta$ to $l_2$ forms a region $R_2$, it should be possible to embed two logos by choosing $l_1$, $l_2$ and $\delta$ appropriately. Note that a gap or a buffer of $\delta$ must be kept between adjacent ranges of variances inorder to avoid block variances moving from one range to another due to change in variance caused by embedding. Ranges for variances must be chosen in such a way that the number of blocks falling in those ranges is equal to the logo sizes.

7.7.2 K-mean Based

Since the center pixels of image blocks can be obtained at various regions, it should be possible to embed multiple watermarks. For example, if a set of center pixels of image blocks are captured in a region $R_1$ forming a set one, and another set of center pixels of image blocks are captured in a region $R_2$ forming a set two, it should be possible to embed two logos. Using k-means on feature vectors of pixel points for segmenting
watermarked images might end up in wrong region recovery, and hence lead to synchronization loss. Voronoi diagram as used in [72] is a favorable one for segmenting watermarked images, because the regions would not change due to watermark embedding. This is because the centroids on pixel positions, rather than the pixels values, are used as keys.

7.7.3 A Comparison between Variance and K-means Based Methods

Our experiments show both the methods perform equally well. The first method involves computation of variances of each block, finding the ranges for block grouping, and finally embedding a logo in each group of blocks. The second one involves segmenting the image into various regions using k-means and choosing each region blocks for embedding a logo. To decide on performance level of the above two methods, one has to take the following into consideration.

1. Amount of computation involved

2. Security level

Figure 7.2 shows the two logos that are embedded in Peppers image, choosing variances of image blocks at two different ranges and the same logos extracted from the watermarked image.

![Figure 7.2: (a) and (b) are embedded logos and (c) and (d) are extracted logos](image)

Figure 7.3 shows the three logos that are embedded in Peppers image after the image has been segmented into three regions by kmeans algorithm using the centroids (23,88), (115,39) and (93,162) trained on image pixel locations. The segmented regions and the three logos that have been recovered are also shown.
7.8 Security Analysis

7.8.1 Block Size as a Key

Since center pixel of adjacent blocks are chosen for embedding, block sizes could be chosen as keys. By choosing varying size blocks for embedding, the security of the scheme could be enhanced.

7.8.2 Seed of PRNG as a key

To accomplish randomization of pixel selection, a pseudo random number generator (PRNG) could be used with a distinct seed for each image.

Randomize Pixel Selection within a Block

Instead of choosing the center pixel of image blocks for embedding a binary bit of a logo, an arbitrary pixel in a block could be chosen. This randomization of pixel selection will avoid all the attacks proposed for Yeung-Mintzer scheme.
Randomize Block Selection within an Image

Instead of choosing the blocks in a raster scan of an image for embedding binary bits of a logo, an arbitrary block within the image could be chosen. This randomization of block selection will avoid all the attacks proposed for Yeung-Mintzer scheme.

7.8.3 Binary Look Up Table as a Key

It has already been proposed that if the embedded pixel points of two or more images with a same logo are known, the binary look up table can easily be guessed. Therefore, major security of our scheme must come from hiding the embedded pixel points rather than hiding the binary lookup table. Unlike our scheme in [9], we do not need alternating zeros and ones in the binary look up table to reduce the synchronization loss during recovery. That is, there is no major restriction on the type of LUTs. As in Yeung-Mintzer scheme, as long as the LUT has randomized zeros and ones, this scheme should perform well.

We did experiments to verify how much watermarked images and extracted logos would change from the original ones for different sequences of 0’s and 1’s in the look up table. As expected highest alternating look up tables (ie. 1010... or 0101...) produced the best PSNR values for watermarked images. This was because the highest alternating look up tables produced the least changes in pixel values during embedding. (Table 7.2 and Figure 7.4)

<table>
<thead>
<tr>
<th>LUT</th>
<th>PSNR (Image)</th>
<th>PSNR (Logo)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>57.995</td>
<td>Infinity</td>
</tr>
<tr>
<td>010101...</td>
<td>65.035</td>
<td>Infinity</td>
</tr>
<tr>
<td>101010...</td>
<td>65.208</td>
<td>Infinity</td>
</tr>
<tr>
<td>001100...</td>
<td>61.271</td>
<td>Infinity</td>
</tr>
<tr>
<td>110011...</td>
<td>60.902</td>
<td>Infinity</td>
</tr>
<tr>
<td>000111...</td>
<td>57.043</td>
<td>Infinity</td>
</tr>
<tr>
<td>111000...</td>
<td>53.175</td>
<td>Infinity</td>
</tr>
</tbody>
</table>

Table 7.2: Look up tables, PSNR values of watermarked image and extracted logo

7.8.4 Centroids as Keys

In our region based method, there can be several regions chosen depending on the number of centroids used. The locations of those centroids can be numerous. The
number of centroids determines the number of logos that could be embedded. The locations of those centroids determine the regions in which the logos are embedded. Since the regions chosen are arbitrary, the centroids of those regions can be used as keys.

### 7.9 Size of Key Space

#### 7.9.1 Block Sizes as Keys

If the image size is $m \times n$, the number of blocks that could be chosen can range from 1 to $m \times n$, assuming the size of blocks range from $1 \times 1$ to $m \times n$. If we consider only square blocks of size $p \times p$, the maximum number of square blocks within an image of size $m \times n$ will be equal to $m/p \times n/p$. In this case, the size of the key space will be equal to $n$ (if $n < m$) or $m$ (if $m < n$).

#### 7.9.2 Seed of a PRNG as a Key

**Arbitrary Pixel within a Block**

Suppose equal size blocks are chosen for embedding, the number of possible choices for choosing a pixel within a block of size $p \times p$ will be equal to $p \times p$. Therefore, the size of the key space will be equal to $(p \times p)^{m/p \times n/p}$. To minimize the size of the key space for an image of size $m \times n$, an appropriate value for $p$ must be chosen.
Arbitrary Block within an Image

Assuming the block sizes are $p \times p$, there can be maximum of $m/p \times n/p$ blocks in an image of size $m \times n$. Suppose a logo of size $m/p \times n/p$ is to be embedded, choosing the blocks in a random order, the size of the key space will be equal to $(m/p \times n/p)!$.

7.9.3 Binary Look Up Table as a Key

There are altogether $2^{256}$ possible look up tables. Out of these tables, The best one for the least modification of the images are 0101..., and 1010.... If the maximum number of consecutive 0's and 1's are kept within 3, there will be several possible tables to choose from.

7.10 A Comparison Between Bhattacharjee’s Scheme and Ours

Bhattacharjee’s scheme some what resembles our multiple watermaring scheme and hence we give a comparison between our scheme and theirs.

1. Their scheme uses Maxican-hat wavelet for feature point detection, whereas ours uses the center pixels of image blocks for feature extraction.

2. In their scheme, Voronoi regions are formed choosing the feature points determined as centroids. In our scheme, Voronoi regions are determined based on centroids trained on arbitrary initial points.

3. Spread spectrum watermarking has been recommended for marking each region in their scheme. We use Yeung-Mintzer scheme for embedding marks on center pixels of image blocks. We use polarity of center pixels as additional information in embedding.

4. They do not have sufficient explanation for choosing the feature points as centroids of Voronoi regions. We choose arbitrary points as centroids of Voronoi regions.

5. Their main reason for watermarking is copyright protection, thus robustness plays a vital role than fragility. Our main reason for watermarking is authentication, thus fragility plays a vital role than robustness.
6. Since their scheme is designed for copyright protection, it has not been made key dependent. Since ours is for authentication, it has been made key dependent by hiding the pixel points chosen within each block as well as the blocks chosen for embedding.

7. In their case, multiple spread spectrum can be embedded by choosing each one for each region, whereas in ours, multiple logos can be embedded by choosing each logo for each region.

7.11 Performance Evaluation

To evaluate the performance of our scheme, after embedding the logo, we performed the following attacks on the watermarked image: JPEG compression at quality level of 90, blurring, embossing, horizontal flipping, oil-painting, rotation to one degree, vertical flipping, sharpening, cutting-1, cutting-2, copy and paste-1, copy and paste-2. Unlike our scheme in [9], we were able to recover the embedded logo in a distorted form from the watermarked images for all the attacks mentioned above. This was because the size of the image blocks and the size of the image remained the same during the process of embedding and extraction. The recovered logo for all attacks mention above are shown in Figure 7.6. Some geometric attacks on Peppers image and the recovered logos for those attacks have also been shown in Figures 7.5 and 7.6 to illustrate the localization of manipulation. The recovery of logos from all attacked images and the localization of manipulations show the superior performance of our scheme.

Figure 7.5: (a) cutting-1 (b) cutting-2 (c) copy and paste-1 and (d) copy and paste-2 attacks on Peppers image

7.12 Conclusion

In this chapter, we have proposed a new fragile watermarking scheme which embeds the marks on the center pixels of image blocks based on the polarity of the pixels. The
Figure 7.6: Logo recovered from (a) JPEG compression at quality level 90 (b) blurring (c) embossing (d) horizontal flipping (e) oilpainting (f) rotation (g) vertical flipping (h) sharpening (i) cutting-1 (j) cutting-2 (k) copy and paste-1 and (l) copy and paste-2

polarity of these points are in tact even after embedding the marks unless the image has been tampered with. This scheme prevents the two main attacks proposed for Yeung-Mintzer’s scheme. The security level of the scheme and the possible extension to multiple watermarking scheme are also investigated.
Chapter 8

Conclusion

Authentication to images can be provided either by annexing a tag to the image or by embedding a secret code or watermark into the image itself. Because of the importance of JPEG compression, providing authenticity to compressed decompressed images has also become essential. Since appending tags to images, in order to provide authenticity is subject to tag removal attacks, embedding watermark into images has become an alternative method for providing authentication.

8.1 Proposed Schemes

The initial goal of this work was to study the existing compression surviving authentication and watermarking schemes, and then to propose elegant hash functions and fragile watermarking schemes. The hash functions we have proposed extract image features that survive acceptable level of JPEG compression, using basic statistical measures, hypothesis of testing, k-mean segmentation, and wavelet or discrete cosine transform. The fragile watermarking schemes, we have proposed embed watermark either on feature points extracted using wavelet or discrete cosine transform on image blocks based on polarity of pixel points.

For all proposed schemes, the security level against possible attacks, efficiency in terms of computation, performance in terms of different statistical or filtering techniques, and the possible improvements or extensions have been explored.

8.2 Open Problems

All our schemes have been experimented with not more than ten images. To rely on these schemes, each scheme must be experimented with several images. The security level of some of the schemes are not investigated well and therefore needs some more work. In particular, the possible attacks on all our statistical based hash functions
are not explored. The schemes which use transformation have only been experimented with four different wavelets. There are numerous wavelets to explore with. In addition, our wavelet based scheme does not sub-sample wavelet co-efficients as described in JPEG 2000. This may be a cause for inefficiency in terms of quality of feature extraction. Although our proposed hashing methods tolerate some level of compression, we could not devise methods for giving compression tolerance to our fragile watermarking schemes. That is to make our schemes semi-fragile. If this had been accomplished, our watermarking schemes for authentication would have become elegant and sophisticated.
Appendix A

Compression and Encryption

This chapter deals with four different areas in combined compression and encryption. Section A.1 describes the compression scheme based on adaptive Huffman coding and the possible steps involved in securing this scheme. More specifically, this has been done by providing protection mechanisms to known attacks on adaptive modelling. Section A.2 reviews all the MPEG Encryption schemes proposed so far. Section A.3 investigates the compression performance of JPEG encryption based on permutation and proposes an alternative to improve on compression performance. Section A.4 proposes an extension to the already proposed scheme of deciphering a file that has been Huffman coded with three symbols but not otherwise encrypted.

A.1 Securing Adaptive Huffman Coding

A.1.1 Introduction

Data compression reduces storage requirement of data, and allows efficient use of channel capacity. Entropy coding algorithms such as Huffman coding [14] and arithmetic coding [15] are not only used for lossless compression but also used as part of lossy compression algorithms such as JPEG (Joint Photographic Experts Group) [26] which is a widely used standard for image compression. In both algorithms there is a model that captures the statistical properties of the input message. A zero-order model consists of a relative frequency table for the input. Higher order models capture higher order statistics and context information. We only consider zero-order model. In static systems the model remains constant during the encoding process, while in dynamic ones the input symbol updates the model too.

Adding encryption to compression algorithms is an attractive proposition which could result in combined security and compression and so more efficiency. The main challenge is to design secure systems without degrading compression performance. A
number of proposals that combine encryption with compression systems have been published in recent years \[59, 60, 61, 63\]. Entropy coding algorithms such as Huffman coding \[14\] and arithmetic coding \[15\] are not only used for lossless compression but also used as part of lossy compression algorithms such as JPEG (Joint Photographic Experts Group) \[26\] and so adding security to them will result in securing the larger systems too. Witten and Cleary \[24\] proposed a method of adding encryption to arithmetic coding. However number of attacks were published against their proposed systems \[22, 25\] and their extensions, which clearly showed designing secure compression systems without drastically affecting compression ratio is not an easy task. In this section we propose a secure compression system based on dynamic Huffman coding.

**Huffman coding** is an optimal compression algorithm that in its simplest form uses probabilities of characters in the input sequence as a model for input source, and produces codewords whose lengths are inversely proportional to their probabilities. In *Dynamic Huffman Coding* (DHC) the model is updated and the codeword representing a symbol changes with its position in the input stream. Following Witten and Cleary approach, we will use the model as the key information and obtain a secure compression system.

We show that the resulting system is vulnerable to attacks which have been already proposed against secure arithmetic coding schemes, and propose a system that protects against these attacks.

The rest of the chapter is organised as follows. In the next section we review DHC, and in Section A.1.3 we describe DHC encryption scheme and examine possible attacks. Section A.1.4 proposes an encryption scheme that protects against the attacks and Section A.1.5 gives the results of our experiments. Section A.1.6, concludes the chapter.

### A.1.2 Dynamic Huffman Coding

Dynamic Huffman coding was independently proposed by Faller \[16\] and Gallager \[17\] and considerably improved by Knuth \[18\] and Vitter \[19\]. The algorithm uses a *code tree* which is a weighted binary tree capturing the statistics of the input stream and forming the coder’s model. Each leaf of the tree corresponds to one alphabet symbol and the weight of the leaf represents the current frequency of that symbol in the input stream representing the weights of each alphabets. Two nodes with a common parent in a code tree are called *siblings*. The weight of an internal node is the sum of the weights of its siblings. Nodes are numbered in increasing order starting
from the bottom left node to bottom right node, followed by the nodes in the layers above again from left to right, until the root node is reached. A binary tree is said to have sibling property if listing the nodes using the above ordering results in a non-decreasing sequence. Gallager [17] proved that a binary prefix code is a Huffman code if, and only if, the code tree has the sibling property. In a dynamic Huffman tree the structure will be maintained in such a way that sibling property is preserved.

To encode an input symbol, first the the current Huffman tree is used to generate the codeword and then the weight of the leaf corresponding to the incoming symbol is updated. This update will flow through the whole tree. If because of updating of the weights, the sibling property is violated a new round of tree updates that involves the exchange of the nodes and their corresponding subtrees, will be applied to reinstate the property. Details of the update can be found in [19].

In implementing the above algorithm, a halving process is introduced. Halving is invoked if the weight of the root node reaches $W_{max}$ which is the maximum possible weight of a node. In this case the weight of each node will be replaced with half its original value. This ensures that the tree structure is not changed and model statistics (probabilities) are also intact. For example if a 16-bit unsigned integer is used for storing $W_{max}$, weights up to $2^{16} = 65535$ can be accommodated.

### A.1.3 Huffman Coding Encryption Schemes

Static Huffman uses a fixed codebook. To add security one can use the codebook as the key. However the resulting system will be vulnerable to simple chosen plaintext attack and hence will not be secure. Using Dynamic Huffman coding is more promising as the codebook is not fixed.

Adaptive Huffman coding encryption scheme is further motivated by the following observations [24].

1. Model for DHC with reasonably large alphabet set (for example, 8 bit per symbol) would be very large and so the key space would be enormous.

2. An adaptive model depends on the entire transmitted text and finding the key would require tracking the changes to the model through the whole text.

The system may work in one of the following two ways [24].

*Scheme 1:* The key is the initial tree which will be securely sent to the decoder. The secret information includes the shape of the tree and the set of weights attached to the nodes and leaves.
Scheme 2: The initial model is public. Encoder and decoder share a secret key sequence which is run through their encoders and updates the code tree. The output is suppressed. At the end of the string the tree has a structure and weights that is unknown for outsiders.

Security of the schemes

The first step in evaluating security of a system is to find the size of the key space. In DHC encryption schemes 1 and 2, the key is the initial tree and the secret pseudorandom string, respectively. In the former case the size of the key space is the number of all possible trees. Assuming there are \( n \) symbols in the alphabet, the total number of possible trees is obtained as follows. The number of binary trees with \( n \) leaves is \( \binom{2n-1}{n} \), known as Catalan number [21]. The number of possible Huffman trees with \( n \) distinct symbols, that is the number of possible keys for Scheme 1, will be

\[
\frac{n! \binom{2n-1}{n}}{n} = \frac{(2n - 1)!}{(n - 1)!}
\]

The size of the key space for Scheme 2 is \( 2^\ell \) where \( \ell \) is the length of the secret string.

It is important to note that the size of the key space only gives an upper bound on the level of security. To have a realistic assessment of the security of the system possible attacks on the system must be considered.

Attacks on Huffman Coding

Ciphertext only attack on static Huffman codes with three symbols has been analysed by Mohtashemi [20]. Extending this analysis to higher number of symbols, for example 256 symbols, and dynamic models is a daunting task. Moreover such study will be mainly of theoretical interest because the attack model is very restricted and in practice the adversary is much more powerful. In the following we extend two attacks originally proposed on adaptive arithmetic coding encryption schemes to DHC encryption schemes.

Bergen and Hogans (BH) attack for adaptive Huffman

Bergen and Hogan [22] proposed an attack on adaptive arithmetic coding encryption scheme. The attack is chosen plaintext and allows the attacker to feed plaintexts of his choice to the encoder and modify the model into a known form. That is, it uses the adaptiveness of the system to direct into a known state. Once the model is known the attacker can decrypt the communication.

The attack scenario is as follows.
1. The system is a DHC encryption scheme 1 or 2.
2. Attacker can send any sequence of symbols to the encoder and obtain the output.
3. Attacker has a decoder.

The attack proceeds as follows.

1. The attacker sends a long string of a single symbol \( \phi \) to cause halving. If the string is chosen long enough, with enough number of halving occurring \( \phi \) eventually moves to the top right hand side of the tree and frequencies of all other symbols become 1. If more \( \phi \) are sent so that another halving occurs the frequency of \( \phi \) will become a constant after each halving. We refer to this state of the tree as *synchronised state*. In the synchronised state inputing each extra \( \phi \) will produce a constant output of 1. At this stage the attacker does not know the positions of other symbols in the tree.

2. If the attacker sends a sequence of a second symbol and observes the output, because of the sibling property of the coder, the symbol will migrate towards the right top of the tree. If halving occurs during this process, already ordered symbols will not be affected. When the encoder starts emitting the first 01 the attacker will know that the second symbol has reached its final position in the tree. The attacker repeats the procedure for the third and fourth symbols until all the symbols in the tree are ordered.

The code tree will look as follows.

![Code tree](image)

Figure A.1: Code tree

Now the model can be verified by decoding a message.

To protect against this attack a proposal by Bergen and Hogan can be adapted. The basic approach is to change the maximum frequency, \( W_{\text{max}} \), using a synchronous pseudorandom sequence in the encoder and decoder. Since halving occurs when \( W_{\text{root}} = W_{\text{max}} \) the halving points become unpredictable. This randomization might reduce the
compression rate and cannot prevent the model being modified. Lim, Boyd and Dawson 
[25] suggest regular resetting of the model during encoding a message which discards 
the model modified by an attacker.

Lim, Boyd and Dawson Attack

Bergen and Hogan attack does not find the key. Lim, Boyd and Dawson showed 
[25] a chosen plaintext attack on arithmetic coding encryption that discovers the key. 
The attack can also be extended to DHC encryption scheme to find the code tree. The 
basic idea is to send single character to the encoder and observe the output which in 
the case of DHC encryption reveals the position of the corresponding leaf. The encoder 
is reset to its initial state and the next symbol is sent to the encoder. In this way the 
position of all symbols and hence the structure of the code tree will be known however 
the weight of the nodes cannot be determined.

To protect against this attack, the system must not be resettable: that is the effect 
of the incoming symbols should not be removable from the model.

A.1.4 Securing Dynamic Huffman Encryption System

In the following we describe two methods of providing protection against BH attack, 
which is the main attack on DHC encryption system. The first method is masking the 
output such that the adversary cannot distinguish the state of the model by observing 
the output, and the second one is by making the system semi-adaptive.

1. Masking the output of an adaptive Huffman encoder

As noted earlier the basic approach in BH attack is to flood the model with a 
particular symbol and by observing the output to determine the state of the symbol in 
the code tree. To prevent the attack we can corrupt the observation data used by the 
advantage.

Two possible methods are i) buffering the output that removes the symbols bound-
daries and makes the input output relationship more complex, and ii) permutation that 
hides the repetitive sequences that mark synchronised state of the model.

Buffering masks the separation between each codeword and prevents the attacker 
from knowing the exact beginning/ending of a codeword. We assume a 1 byte buffer 
is used in the output and describe masking through an example.

When the model is flooded with a sequence of the same symbol, the system will 
ultimately produces a string of \( a_1 = 1111111 \). Knowing \( n \), the number of symbols used 
to reach this stage, the attacker can send another sequence of length at most \( n \) until 
a sequence of \( a_2 = 01010101 \), is obtained. This process is continued until the attacker
can synchronise the whole tree. For the third symbol, the buffer will be filled with multiple copies of 001 which depending on the starting point can produce one of the following blocks: \( a_3 = 00100100, a_4 = 10010010, a_5 = 01001001. \)

If we only consider the top 3 levels of the code tree, masking must be applied to \( A = \{a_1, a_2, \cdots, a_5\} \). A simple masking is to replace elements of \( A \) with another set of blocks \( B = \{b_i, i = 1, \cdots, 5\} \). This mapping will be secret and its inverse will be used in the decoder. In this case to avoid erroneous decoding, it is necessary to map back elements of \( B \) to elements of \( A \). So, masking will consist of a hidden substitution of elements of \( A \) with \( B \), and those of \( B \) with \( A \). Let \( a \) and \( b \) denote the sizes of \( A \) and \( B \) respectively, and \( m \) denote the size of the alphabet set. Then \( B \) can be chosen in \( \binom{m}{a} \) and the masking table may be defined in \( (a!)^2 \times \binom{m}{a} \). However although masking can hide the exact form of an output block but cannot hide the repetitive nature of the output sequence, as it replaces a sequence of \( a_i \) with a sequence of \( b_j \)'s. This means that in this basic form masking cannot prevent detection of the synchronised state.

A possible solution is to use more than one substitution table and use a pseudo-random generator to alternate between them. To minimise the amount of storage, one can use a master table \( T \) and a number of other tables \( t_1, \cdots, t_u \) which are obtained by permuting the rows of \( T \). If \( T \) has \( \ell \) rows, then the tables can be chosen in \( \binom{\ell}{u} \) ways. Combined with the number of ways of constructing \( T \), the result will be a huge key space which makes the exhaustive key search infeasible.

By increasing the length of the buffer and considering more levels of the trees the size of the key space can be further increased.

2. Semi-adaptive Time Variant Modeling

Barbir proposed [23] a semi-adaptive modeling approach for combined compression and encryption using arithmetic coders. The basic idea is to introduce some randomness into model probabilities. This is achieved by performing updates at random intervals specified by the output of a pseudorandom sequence.

Using a similar approach in DHC encryption can prevent BH attack. The system works as follows. The encoder works as normal: it reads the input, produces an output and updates the code tree. After \( n_i \) input symbols where \( n_i \) is the output of a pseudorandom generator, let \( f_1^i, \cdots, f_m^i \) denote the frequencies of the \( m \) symbols in the input alphabet. At this stage a sequence of length \( n_i' \), where \( n_i' \) is determined by the pseudorandom generator, is used to update the model without producing any output. To reduce the effect of this randomisation on compression ratio, the number of the \( j^{th} \) symbol in the update string will be chosen proportional to \( f_j^i/n_i \).
This ensures that the model will match the input data and the input probability distribution will not change and the compression performance is not affected.

A.1.5 Experiment

We encrypted the Lenna image using a JPEG implementation which uses adaptive Huffman coding for its entropy coding stage. Next the image was encrypted with the proposed DHC encryption scheme. We decoded the encrypted image without decrypting it (Figure A.2(b)) and then decoded the encrypted image with a wrong key (Figure A.2(c)). These experiments showed that compression ratios do not drop after adding encryption. Moreover using an incorrect key does not reveal any partial information.

A.1.6 Conclusion

We showed that a model-based approach to combined compression and encryption using adaptive Huffman coding results in a system that is vulnerable to attacks known on arithmetic coding encryption schemes. We proposed methods of preventing against these attacks and gave the results of incorporating the system into JPEG compression.

![Figure A.2: (a) Original Lenna (b) Lenna after encryption and (c) Lenna after decryption with a wrong key](image)
A.2 Critical Analysis of MPEG Encryption Schemes

A.2.1 Introduction

In this review secrecy of multimedia data is considered and compared. Secrecy of multimedia is very important for various reasons. For instance, it is desirable in a video-on-demand systems that only those computer users who have paid for the service can access those videos or movies. In a video conferencing system, only those members of the conference can watch the video. Vast amount of data of these multimedia applications put great burden on the encoding process. Encryption or decryption will aggravate this problem. How to encrypt and decrypt the vast amount of data efficiently is an important issue.

Digital image processing techniques are used to enhance the quality of image and to reduce the data needed to represent an image with out visually affecting the quality of image (compression). Cryptographic techniques are used to scramble images so that an adversary could not obtain the original image with out knowing the secret key.

MPEG is the industrial standard for video compression. There are three standards MPEG1, MPEG2, and MPEG4 targeted for different applications. MPEG1 has been widely used. It targets VHS quality video (320x240) with a CD quality audio at the speed of 1.5Mbits/s. It consists of 3 parts video, audio and system to control interleaving of systems.

A.2.2 Incorporation of Compression and Encryption

Due to the spatial property of digital images it is unnecessary to treat digital images as bit stream and encrypt them bit by bit. New lightweight cryptographic algorithm should be invented. We give a list of desired properties that an ideal digital image encryption algorithm should have.

1. Due to the high information rate of the digital image data, the computational overhead introduced by the encryption procedure and decryption procedure should be as low as possible.

2. encryption procedure should not decrease the compression rate.

3. The encryption algorithms should be robust against common image processing techniques such as image enhancement, image restoration, etc.,

4. The encryption and decryption of the digital images should not affect the quality of the original images.
A.2.3 MPEG Encryption Algorithm

Naive Algorithm

The most straight-forward method is to encrypt the entire MPEG stream using standard encryption methods such as DES [57]. This is called the naive algorithm approach. Naive algorithm treats the MPEG bit-stream as the traditional text data and does not use any of the special structure.

This is supposed to be the most secure MPEG encryption algorithm. This is because so far there is no effective algorithm to break DES, especially double or triple DES which uses two or three different keys.

This encrypts entire MPEG stream. It is slow especially when we use double or triple DES to achieve the top security level. The size of the encrypted stream does not change because most standard encryption algorithms preserve the size.

Selective Algorithm

There are several proposals which use the features of MPEG layered structures [58]. These algorithms all fall into the category of Selective Algorithm.

The encryption and decryption of video at rates of 30-60 Mbps is not possible with the currently available encryption algorithms. In anticipation of the great need for a security mechanism for real-time video applications emerging with the implementation of high bandwidth networks, a new security mechanism is proposed: Aegis [59]. In order to efficiently utilize available bandwidth and storage capacity, it is expected that digitized video will be compressed using a standard compression algorithm.

Aegis exploits the great sensitivity of compressed video. The approach used by Aegis is the encryption of the I frames for all MPEG groups of frames in a MPEG video stream. The choice of encrypting the I frames is based on the great significance of the intraframe in the decompression of a MPEG stream. B and P frames represent only translations of the picture information found in adjacent I frames; therefore, the encryption of I frames renders them useless. Further more, the intentional corruption of the stream has a serious impact on the outputs of the inverse DCT function during decoding. The recovery from such corruption is practically impossible.

Partial encryption of the MPEG video stream, provides some level of security because it presents two challenges to the ambitious network intruder. When portions of the MPEG stream are encrypted, the MPEG stream does not confirm to the standard MPEG stream layered structure, and consequently, it is impossible to identify frames,
group of frames, or the encrypted I frames. The network intruder must first separate the encrypted portions of the stream, and then still faces the complexity of the encryption algorithm.

In addition to the encryption of I frames, Aegis also encrypts the MPEG video sequence header. The sequence header contains all of the decoding initialization parameters such as the picture width, height, frame rate, bit rate and buffer size. The encryption of the sequence header, also conceals the MPEG identity of the stream and makes the MPEG video stream unrecognizable. In order to further conceal the MPEG identity of the stream, the ISO end code (last 32 bits of MPEG) is also encrypted. Aegis uses DES for the encryption process.

Agi and Gong have shown that great portions of the video are visible partly because of inter-frame correlation and mainly from unencrypted I blocks in the P and B frames. Therefore, encrypting only I frames does not provide a satisfactory security level. In order to provide better security level, in addition to encrypting I frames, all I blocks in P and B frames must also be encrypted. Identifying the I-blocks will introduce overhead. Encrypting only I frames can save 30-50% of encryption/decryption time. Size does not increase.

Agi and Gong have also suggested to increase the frequency of I frames to enhance the security. It has the main drawback of increasing the length of string and consequently the encryption time.

A.2.4 Zig-Zag Permutation Algorithm

The purpose of encrypting text information is to prevent an adversary without the secret key from obtaining the information. In this instance, the content of the information is either known completely or it is unknown. But there are two levels of security to digital images.

1. Scrambled image has poor image quality compared with original image, but the content of the original image is visible to the viewer - obscured image

2. Scrambled image is not comprehensible - incomprehensible image

There are some special properties unique to the JPEG and MPEG encoding. After DCT and the quantization procedure, the coefficients of every 8x8 block has the following properties.

1. All coefficients are in the range of [0, 255] which can be represented by 8-bit binary string.
2. The amplitude of the DC coefficient is much larger than that of every AC coefficient.

3. Big portion of AC coefficients are zeros.

The basic idea of Zig-Zag Algorithm is that, instead of mapping the 8x8 block to a 1x64 vector in "Zig-Zag" order, it uses a random permutation list to map the individual 8x8 block to a 1x64 vector.

Following experiments are conducted by Tang [60].

1. DC coefficient is mapped to the first element in the 1x64 vector and the rest of the elements are permuted. $\rightarrow$ Obscured image

2. DC coefficient of every block is set to zero or a fixed value between 0 and 255 and rest of the elements are permuted. $\rightarrow$ Obscured image

3. DC coefficient is mapped to any other position other than the first position in the 1x64 vector, and the rest of the elements are randomly permuted $\rightarrow$ Incomprehensible image

4. AC63 coefficient is set to 0 $\rightarrow$ Degradation is negligible

5. Split the DC coefficient into two parts, first part remain in the same position, the second part is substituted for AC63 and randomly permute the list $\rightarrow$ Incomprehensible image

The basic Zig-Zag Permutation Algorithm is vulnerable to the ciphertext only attack. The attack is based on the fact that none-zero AC coefficients are gathered in the upper-left corner of the I block. Statistical analysis which count the number of non-zero DC and AC coefficients from all blocks in an I frame was conducted by Qiao and Nahrstedt [61]. The results show

1. DC coefficients always have the highest frequency of non-zero occurrence

2. The frequency of AC1 and AC2 are among the top 6.

3. The frequency of AC3 to AC5 are among top 10.

The other method is to use so-called binary coin flipping sequence together with two different permutation lists. For each 8x8 block, a coin is flipped. If it is a tail, the permutation list 1 (key1) is applied to the block; if it is a head, the permutation list
2 (key2) is applied to the block. Key1 and Key2 are the secret keys. (L.Tang) This method is subject to known plain text attack. The idea is to select the key that has the tendency to gather AC coefficients in the upper left corner.

A.2.5 Video Encryption Algorithm

Qiao and Nahrstedt [61] have studied the statistical properties of MPEG stream. Their study was based on the fact that both compression and encryption remove redundant information. This means that after encryption or compression the bit stream will have a more uniform distribution. They have chosen MPEG stream in byte-by-byte fashion for following reasons.

1. Easier to handle
2. Unlike text data single bytes are meaning less
3. Randomness is introduced at the byte level (this reason is debatable because the randomness is firstly introduced at I frames, secondly at P and B frames when motion compensation prediction are made, and finally at entropy/run-length coding level)

For experimental purposes they have chosen:

1. MPEG video with different features; video with rapidly changing background and little changing background for experiments.
2. First and second half of the byte stream or a randomly chosen half of the byte stream.
3. Frequency distribution of digrams among all 256x256 possible pairs.

All these experiments have produced uniform distributions.

To measure the periodicity of the MPEG streams the following experiment was carried out:

1. Divide the I-frames into equal length chunks (1/8 or 1/16 of the I frame)
2. Define random variable x as the number of occurrences of highest frequency pair in one chunk
3. Calculate P(x > 1)

For all tested video schemes this experiment has produced that the chance of repeating even the highest frequency diagram within a 1/16 chunk is less than 0.03.
Algorithm

1. Choose odd-numbered bytes and even numbered bytes.

2. Xor the two streams \( a_1, a_2...a_{2n-1} \) Xor \( a_2, a_4...a_{2n} \) to produce the stream \( c_1, c_2...c_n \).

3. Choose an encryption function \( E(DES) \) to encrypt \( a_2, a_4...a_{2n} \) resulting ciphertext has the form \( c_1, c_2...c_n \ E(a_2, a_4...a_{2n}) \)

KeyM

To enhance security instead of using odd and even list randomly generated 128 bit key KeyM with 64 0’s and 64 1’s is used.

Keyi

Frame I has 64 bytes. Each 1/16 chunk has 4 bytes. 4 bytes have 32 bits. 1/2 frame has 8 chunks - key1—key8, each key is a random permutation of \((1,...32)\). Length of keyi is \(32 \times 5/8 = 20\) bytes - total length of 8 keys is \(160 = 8 \times 20\) bytes

KeyF

To change the pattern of choosing even and odd list, each frame is assigned a 64-bit key. Key is a permutation of \((1,...,16)\) where each number is a 4 bits.

KeyE

This is the key for DES.

This scheme is immune to known-plaintext attack because the key will be changed for each frame. The best way to perform ciphertext only attack is to use frequency analysis. The attack is based on finding a pair \((a,b)\) such that a xor b = c. From the statistical analysis since the highest pair frequency is 10-4 the chance of guessing these pairs is very remote and hence the ciphertext only attack is very difficult.

The author claims VEA is faster (47%) than DES because DES is used on partial bit stream.

A.2.6 Sign-bit Encryption

Original Scheme

This sign-bit encryption scheme [63] is a selective encryption scheme which operates on the sign bits of DCT coefficient of a MPEG compressed video. The key \( k = b_1b_2...b_m \)
is a randomly generated bit stream of length $m$. Encryption function can be given as follows:

$$E_k(S) = \ldots (b_1 XOR s_1) \ldots (b_m XOR s_m) (b_1 XOR s_{m+1}) \ldots (b_m XOR s_{2m}) \ldots$$

Where $s_1 s_2 \ldots s_m s_{m+1} \ldots s_{2m} \ldots$ are all of the sign bits of DC and AC coefficients. The encryption operation randomly changes the sign bits of DCT coefficients. The decryption function $E_k^{-1}$ is the same as the encryption function since $E_k(E_k(S)) = S$. For a key of length $m$ an adversary needs to try $2m$ times in order to find a key.

This scheme is much more efficient than DES because it selectively encrypts the MPEG stream. The authors believe, "The block encryption algorithms developed to secure text data usually use short keys and complex computations. This is because text data has small size and cryptography experts want to make adversaries reveal secret keys computationally impossible. For multimedia applications the reverse is true. It needs long keys to prevent attacks and use simple computation to improve on performance."

Modification

A more efficient way to encrypt MPEG video is to use a secret key randomly changing the sign bits of differential values of DC coefficients of I frames and the sign bits of differential values of motion vectors. It has been stated that encrypting motion vectors is very efficient than encrypting DCT coefficients of B and P frames because it not only change the directions of motion vectors but also change the magnitude of motion vector. This scheme is very efficient in terms of computational complexity because it omits the encryption of AC coefficients altogether. The only disadvantage of this scheme is that because DC coefficients and AC coefficients are related DC coefficients may be derived from AC coefficients for an attack.

A.2.7 Conclusion

In this section, after describing MPEG we have critically analysed all proposed MPEG encryption schemes. The proposed methods have been classified into naive algorithm, selective algorithm, zig-zag permutation algorithm, video encryption algorithm and sign-bit encryption. The advantages and disadvantages of each method have also been discussed.
A.3 Compression Performance of JPEG Encryption Scheme

A.3.1 Introduction

Adding encryption to compression algorithm is an attractive proposition which could result in combined security and compression. The main challenge is to design secure systems without degrading compression performance. A number of algorithms have been proposed to provide security to MPEG [58, 59, 60, 61]. Out of all these algorithms, the one which has been analysed for security and compression performance is the Zig-Zag Algorithm proposed by Tang [60]. However, an attack to this scheme has also been proposed in [61].

Without degrading the picture quality, improving the compression performance is possible, only if the lossless compression such as entropy coding is replaced with a different one. A possible alternative is to use a dynamic Huffman coding scheme instead of a static one recommended by JPEG.

Huffman coding is an optimal compression algorithm that in its simplest form uses probabilities of characters in the input sequence as a model for input source, and produces codewords whose lengths are inversely proportional to their probabilities. In Dynamic Huffman Coding (DHC) the model is updated and the codeword representing a symbol changes with its position in the input stream.

In this section, after reviewing the only JPEG encryption scheme known to date, we examine the compression performance of JPEG which has been applied a Zig-Zag Permutation Algorithm, propose a security enhancement to the current scheme, and suggest an alternative to static Huffman recommended by the JPEG in order to improve on compression performance.

The next section reviews DHC, and Section A.3.3 reviews the only known JPEG encryption scheme. Section A.3.4 describes the Zig-Zag Permutation Algorithm. In Section A.3.5, we examine the compression performance of JPEG which has been applied a Zig-Zag Permutation Algorithm. Section A.3.6 concludes the paper.

A.3.2 Dynamic Huffman Coding

Dynamic Huffman coding was independently proposed by Faller [16] and Gallager [17] and considerably improved by Kntth [18] and Vitter [19]. The algorithm uses a code tree which is a weighted binary tree capturing the statistics of the input stream and forming the coder’s model. Each leaf of the tree corresponds to one alphabet
symbol and the weight of the leaf represents the current frequency of that symbol in the input stream representing the weights of each alphabets. Two nodes with a common parent in a code tree are called siblings. The weight of an internal node is the sum of the weights of its siblings. Nodes are numbered in increasing order starting from the bottom left node to bottom right node, followed by the nodes in the layers above again from left to right, until the root node is reached. A binary tree is said to have sibling property if listing the nodes using the above ordering results in a non-decreasing sequence. Gallager [17] proved that a binary prefix code is a Huffman code if, and only if, the code tree has the sibling property. In a dynamic Huffman tree the structure will be maintained in such a way that sibling property is preserved.

To encode an input symbol, first the the current Huffman tree is used to generate the codeword and then the weight of the leaf corresponding to the incoming symbol is updated. This update will flow through the whole tree. If because of updating of the weights, the sibling property is violated a new round of tree updates that involves the exchange of the nodes and their corresponding subtrees, will be applied to reinstate the property. Details of the update can be found in [19].

### A.3.3 JPEG Encryption Scheme

Although many proposals exist for encrypting MPEG, only a limited amount of research has been done in encrypting JPEG. This may be because JPEG has been considered as a subset of MPEG. Most of the schemes described for MPEG can also be applied to JPEG as well. The only known encryption scheme proposed for JPEG has been given in [64]. Their scheme has considered encrypting DC and low frequency AC coefficients of DCT of each block and the following shortcomings have been noted.

1. If only DC coefficients are encrypted, a good approximation to the original image can be obtained by assigning any value to the DC coefficients.

2. If only DC and low frequency AC coefficients are encrypted, a reasonable approximation to the image can be obtained. This is because the un-encrypted high frequency AC coefficients will partially reveal the edge information of the image.

The above shortcomings have led the authors to propose a spatial domain encryption scheme.
A.3.4 Zig-Zag Permutation Algorithm

The purpose of encrypting text information is to prevent an adversary without the secret key from obtaining the information. In this instance, the content of the information is either known completely or it is unknown. But there are two levels of security to digital images.

1. Scrambled image has poor image quality compared with original image, but the content of the original image is visible to the viewer - obscured image

2. Scrambled image is not comprehensible - incomprehensible image

There are some special properties unique to the JPEG and MPEG encoding. After DCT and the quantization procedure, the coefficients of every 8x8 block has the following properties.

1. All coefficients are in the range of $[0, 255]$ which can be represented by 8-bit binary string.

2. The amplitude of the DC coefficient is much larger than that of every AC coefficient.

3. Big portion of AC coefficients are zeros.

The basic idea of Zig-Zag Algorithm is that, instead of mapping the 8x8 block to a 1x64 vector in ”Zig-Zag” order, it uses a random permutation list to map the individual 8x8 block to a 1x64 vector.

Following experiments are conducted by Tang [60].

1. DC coefficient is mapped to the first element in the 1x64 vector and the rest of the elements are permuted. → Obscured image

2. DC coefficient of every block is set to zero or a fixed value between 0 and 255 and rest of the elements are permuted. → Obscured image

3. DC coefficient is mapped to any other position other than the first position in the 1x64 vector, and the rest of the elements are randomly permuted → Incomprehensible image

4. AC63 coefficient is set to 0 → Degradation is negligible
5. Split the DC coefficient into two parts, first part remain in the same position, the second part is substituted for AC63 and randomly permute the list → Incomprehensible image

The basic Zig-Zag Permutation Algorithm is vulnerable to the ciphertext only attack. The attack is based on the fact that none-zero AC coefficients are gathered in the upper-left corner of the I block. Statistical analysis which count the number of non-zero DC and AC coefficients from all blocks in an I frame was conducted by Qiao and Nahrstedt [61]. The results show

1. DC coefficients always have the highest frequency of non-zero occurrence.
2. The frequency of AC1 and AC2 are among the top 6.
3. The frequency of AC3 to AC5 are among top 10.

The other method is to use so-called binary coin flipping sequence together with two different permutation lists. For each 8x8 block, a coin is flipped. If it is a tail, the permutation list 1 (key1) is applied to the block; if it is a head, the permutation list 2 (key2) is applied to the block. Key1 and Key2 are the secret keys. (L.Tang) This method is subject to known plain text attack. The idea is to select the key that has the tendency to gather AC coefficients in the upper left corner.

### A.3.5 Compression Performance

In this section, we analyse the compression performance of JPEG, if a random permutation has been applied to the zig-zag ordered DCT coefficients. The same experiment had been done for MPEG and the results can be found in [62]. Their findings reveal that zig-zag permutation for MPEG increases the MPEG stream size by as much as forty-six percent. As we have described before, this scheme is also vulnerable to chosen-plaintext attack if the permutations are not randomly chosen. To enhance security we propose applying permutations randomly selected from a set of known permutations after permuting the blocks.

### Experimental Results

We performed experiments on Una, House, Tree, and Lenna images. First we looked at the compression ratios on all four images (Table A.1) for three different quality levels without permutation. We did this experiment in order to compare against other experiments.
We also experimented the compression ratios when the permutations were randomly selected from a list of four different permutations to permute the zig-zag ordered DCT coefficients for each block (Table A.2). What we observed was a maximum of three percent compression drop compared to JPEG without permutation. (Compare Table A.1 and Table A.2)

An adaptive Huffman coding algorithm was incorporated into JPEG instead of a static one, which had been recommended by JPEG, to see how much compression drop may occur (Table A.3). In fact compression rate increased by a maximum of four percent on Una image and by a maximum of three to two percent on all the other images (Compare Table A.1 and Table A.3).

These experiments showed that incorporating adaptive Huffman coding in JPEG achieved a better compression rate than the static one. Replacing static Huffman with a dynamic one in JPEG might compensate for the compression drop that occurred when random permutations had been applied to the zig-zag ordered DCT coefficients.

### A.3.6 Conclusion

In this section, we investigated the compression performance of JPEG when a random permutation had been applied to the zig-zag ordered DCT coefficients. We found that JPEG sequence increased by at most twenty percent compared to MPEG’s forty-six percent. There was a fifty percent performance improvement for JPEG. We also noted a performance improvement in compression when static Huffman coding was replaced with an adaptive one.
<table>
<thead>
<tr>
<th>Image</th>
<th>Compression (ratio)</th>
<th>Quality</th>
<th>PSNR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Una.pgm</td>
<td>1:0.35</td>
<td>3</td>
<td>39.4978</td>
</tr>
<tr>
<td>Una.pgm</td>
<td>1:0.11</td>
<td>25</td>
<td>27.933</td>
</tr>
<tr>
<td>Una.pgm</td>
<td>1:0.15</td>
<td>14</td>
<td>30.7209</td>
</tr>
<tr>
<td>House.pgm</td>
<td>1:0.12</td>
<td>3</td>
<td>25.9796</td>
</tr>
<tr>
<td>House.pgm</td>
<td>1:0.06</td>
<td>25</td>
<td>20.0447</td>
</tr>
<tr>
<td>House.pgm</td>
<td>1:0.07</td>
<td>14</td>
<td>22.3766</td>
</tr>
<tr>
<td>Tree.pgm</td>
<td>1:0.19</td>
<td>3</td>
<td>39.2094</td>
</tr>
<tr>
<td>Tree.pgm</td>
<td>1:0.08</td>
<td>25</td>
<td>30.2431</td>
</tr>
<tr>
<td>Tree.pgm</td>
<td>1:0.09</td>
<td>14</td>
<td>32.3513</td>
</tr>
<tr>
<td>Lenna.pgm</td>
<td>1:0.12</td>
<td>3</td>
<td>38.5824</td>
</tr>
<tr>
<td>Lenna.pgm</td>
<td>1:0.06</td>
<td>25</td>
<td>31.5417</td>
</tr>
<tr>
<td>Lenna.pgm</td>
<td>1:0.07</td>
<td>14</td>
<td>33.6306</td>
</tr>
</tbody>
</table>

Table A.1: Compression ratios for JPEG without permutation

<table>
<thead>
<tr>
<th>Image</th>
<th>Compression (ratio)</th>
<th>Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Una.pgm</td>
<td>1:0.38</td>
<td>3</td>
</tr>
<tr>
<td>Una.pgm</td>
<td>1:0.11</td>
<td>25</td>
</tr>
<tr>
<td>Una.pgm</td>
<td>1:0.16</td>
<td>14</td>
</tr>
<tr>
<td>House.pgm</td>
<td>1:0.13</td>
<td>3</td>
</tr>
<tr>
<td>House.pgm</td>
<td>1:0.07</td>
<td>25</td>
</tr>
<tr>
<td>House.pgm</td>
<td>1:0.08</td>
<td>14</td>
</tr>
<tr>
<td>Tree.pgm</td>
<td>1:0.21</td>
<td>3</td>
</tr>
<tr>
<td>Tree.pgm</td>
<td>1:0.08</td>
<td>25</td>
</tr>
<tr>
<td>Tree.pgm</td>
<td>1:0.10</td>
<td>14</td>
</tr>
<tr>
<td>Lenna.pgm</td>
<td>1:0.13</td>
<td>3</td>
</tr>
<tr>
<td>Lenna.pgm</td>
<td>1:0.07</td>
<td>25</td>
</tr>
<tr>
<td>Lenna.pgm</td>
<td>1:0.08</td>
<td>14</td>
</tr>
</tbody>
</table>

Table A.2: Compression ratios after each block’s zig-zag ordered DCT coefficients have been permuted with one of four different permutations
<table>
<thead>
<tr>
<th>Image</th>
<th>Compression (ratio)</th>
<th>Quality</th>
<th>PSNR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Una.pgm</td>
<td>1:0.31</td>
<td>3</td>
<td>39.38</td>
</tr>
<tr>
<td>Una.pgm</td>
<td>1:0.09</td>
<td>25</td>
<td>27.9374</td>
</tr>
<tr>
<td>Una.pgm</td>
<td>1:0.13</td>
<td>14</td>
<td>30.7239</td>
</tr>
<tr>
<td>House.pgm</td>
<td>1:0.10</td>
<td>3</td>
<td>25.8399</td>
</tr>
<tr>
<td>House.pgm</td>
<td>1:0.04</td>
<td>25</td>
<td>20.3442</td>
</tr>
<tr>
<td>House.pgm</td>
<td>1:0.05</td>
<td>14</td>
<td>22.7807</td>
</tr>
<tr>
<td>Tree.pgm</td>
<td>1:0.17</td>
<td>3</td>
<td>39.1911</td>
</tr>
<tr>
<td>Tree.pgm</td>
<td>1:0.05</td>
<td>25</td>
<td>30.2597</td>
</tr>
<tr>
<td>Tree.pgm</td>
<td>1:0.08</td>
<td>14</td>
<td>32.3642</td>
</tr>
<tr>
<td>Lenna.pgm</td>
<td>1:0.10</td>
<td>3</td>
<td>38.204</td>
</tr>
<tr>
<td>Lenna.pgm</td>
<td>1:0.04</td>
<td>25</td>
<td>31.3555</td>
</tr>
<tr>
<td>Lenna.pgm</td>
<td>1:0.05</td>
<td>14</td>
<td>33.4016</td>
</tr>
</tbody>
</table>

Table A.3: Compression ratios for JPEG with adaptive Huffman coding
A.4 On Breaking Huffman Codes with Four Symbols

In this section, we examine the problem of deciphering a file that has been Huffman coded with four symbols but not encrypted. We show that Huffman code can be even more difficult to crypt analyze as the number of symbols increase from three to four. A detail analysis of the situation for a three-symbol alphabet has already been proposed. We extend this analysis to four- symbol source alphabet. This work is only an extension to the theory developed for three-symbol alphabet at MIT and therefore major part of it comes from [29].

A.4.1 An Overview of the Problem

Huffman’s algorithm for compression was the first one to stimulate the rapid growth in the field of data compression. Adaptive compression schemes that adaptively change to source symbol statistic have been proposed in [18, 21, 28, 27].

The question we try to address is ” How the complexity increases to crypt analyze a message that has not been encrypted when the source symbols increase from three to four. ”

We limit our analysis to Huffman codes with four symbols. We assume the reader knows the basic Huffman algorithm. Suppose for a four symbol source alphabet A, B, C, D if the Huffman coding produces the encoding rule 000, 001, 01, 1, the source string such as DDCBA would be encoded as 1101001000.

We assume that the encoding rule such as above is kept as a private key. We assume the adversary knows the general system: the size of the source alphabets and the fact that Huffman coding is being used.

We note that for a four-symbol source alphabet there are basically only five different Huffman code word sets: $C_0 = 000, 001, 01, 1, C_1 = 00, 010, 011, 1, C_2 = 00, 01, 10, 11, C_3 = 0, 100, 101, 11C_4 = 0, 10, 110, 111$ Thus for a four-symbol alphabet there are five possible encoding rules available- 3 bit secret key is needed to specify which code word set is being used. An additional log(20) bits of secret key can then specify which code words are used for each symbol.

As an example consider the file 1101001000. The crypt-analyst must decide whether it is code word set $C_0$, correspond to a parse such as

$1101001000 \rightarrow DDCBA$

or code word set $C_1$ giving a parse such as
1101001000 $\rightarrow$ $DDBBA$

or code word set $C_2$ giving a parse such as

1101001000 $\rightarrow$ $DBACA$

or code word set $C_3$ giving a parse such as

1101001000 $\rightarrow$ $DABBA$

or code word set $C_4$ giving a parse such as

1101001000 $\rightarrow$ $CBABAA$

Even in this instance her decision is ambiguous between $C_0$ and $C_4$, $C_1$, $C_2$ and $C_3$ are inconsistent with the Huffman algorithm since it gives a frequent source symbol a longer code word.

Some assumptions made on their analysis are

1. When there are more than two symbols of same probability Huffman algorithm must decide how to choose these symbols. Here we assume such ties do not occur.

2. She may know the source probabilities $p_1, p_2, ..., p_n$ either exactly or approximately. Here we assume that the crypt analyst does not know the $p_i$’s.

3. She may know that each source symbol is produced independently of the previously produced source symbols. We call this independent source case.

4. She knows the size $n$ of the source alphabet.

5. There are two variants to the Huffman tree. We say a Huffman tree a right heavy Huffman tree if the sub tree of greater total weight is always made the right sub tree.

In this research we focus on the problem of determining the code word set used to create a Huffman coded file for four-symbol source alphabet as an extension to the analysis for three symbol source alphabets proposed by David W.Gillman, Mojdeh Mohtashemi and Ronald L Rivest [30]. This problem is equivalent to determining the tree forming the Huffman code without determining how the source symbols are attached to the leaves of the tree. We extend the result obtained in [29] for resolving the ambiguity to the case of four symbol source alphabets. Such ambiguity is a blessing to the cryptographer but a problem for a crypt analyst.

### A.4.2 Four Symbol Source Alphabet

In this section we study the problem of breaking a Huffman code for a four-symbol source alphabet using the Cross Product Machine (CPM).
We assume that the source alphabet is \( P, Q, R, S \) and that these symbols have respective probabilities \( p, q, r, s \) in the source file. We also assume \( C_2 \) was chosen by the Huffman algorithm to code the file so that \( p, q, r \) and \( s \) are respective probabilities of 00, 01, 10 and 11 when the file is passed according to \( C_2 \). When the files are passed according to \( C_0, C_1, C_2, C_3 \) and \( C_4 \) the probability distributions are given in the diagram below.

![Huffman Tree Diagram]

Figure A.3: All possible Huffman trees with four symbols

Figure A.3 shows five possible Huffman trees, code words they represent and the probability distribution they take.

We give an extension to the question: "How does a probability distribution on the generator induce a probability distribution on the parser for a particular pair of huffman trees?"

**Markov Chain**

A **Markov chain** is a finite state machine. Emerging from each state are two transition arrows: one emitting a 0 and the other emitting a 1. If we associate the probability \( p_i \) with one of the emerging arrows, then the other arrow is assigned the probability \( 1 - p_i \).

A markov chain generates a string of 0s and 1s by starting at the designated start state, moving along the chain, and outputting 0s and 1s by selecting the appropriate arrows.

**The Cross-Product-Machine**

Given two Markov chains \( M_1, M_2 \) with \( m \) states, where states \( s_1, s_2, \ldots, s_m \in M_1 \) and states \( t_1, t_2, \ldots, t_m \in M_2 \), we define the cross-product-machine \( M \) to be a Markov chain with the following properties:

1. \( M \) consists of \( m^2 \) pair-states.
2. Every pair-state is labeled \( s_i, t_j, 1 \leq i, j \leq m \), where \( s_i \in M_1 \) and \( t_j \in M_2 \).

3. \( M \) has a 0-arrow from one pair-state, \( s_i, t_j \) to \( s_l, t_k \), if
   - in \( M_1 \), there is a 0-arrow from \( s_i \) to \( s_l \) and
   - in \( M_2 \), there is a 0-arrow from \( t_j \) to \( t_k \).

4. Similarly, \( M \) has a 1-arrow from one pair-state, \( s_i, t_j \) to \( s_l, t_k \), if
   - in \( M_1 \), there is a 1-arrow from \( s_i \) to \( s_l \) and
   - in \( M_2 \), there is a 1-arrow from \( t_j \) to \( t_k \).

Given the probability distribution on a set of Huffman codewords (generator) and another set of codewords (parser) with unknown probability distribution, the CPM can be used to induce a probability distribution on the parser.

**Adversarial Source**

In this subsection we assume that the crypt analyst knows only that the file was encoded using the right heavy Huffman algorithm and the source has four symbols. The source is adversarial: that is it generates any stream of symbols in which \( P, Q, R \) and \( S \) appear with frequencies \( p, q, r \), and \( s \) respectively, but the order is arbitrary. Since we assume that right heavy Huffman algorithm uses code word set \( C_2 \), we have \( s \geq r \), \( q \geq p \), and \( r + s \geq p + q \geq 1/2 \).

**Theorem:** Let \( (p, q, r, s) \) be a probability distribution of rational numbers. Then there exist an adversarial four letter source with probability distribution \( (a, b, c, d) \) on the source alphabet which produces an ambiguous file if and only if \( (q - r) \leq (s - p) \leq 1 \)

**Proof:** Assume the file is ambiguous. Frequency of 1’s in file is

\[
q + r + 2s/2(p + q + r + s) = b + c + d/3a + 3b + 2c + d \leq 1
\]

\[
1 + s - p/2 = 1 - a/1 - c - 2d \leq 1
\]

Right hand side takes the value of 1 when \( a = c + 2d \)

\[
1 + s - p \leq 2
\]

\[
s - p \leq 1
\]
\( q - r \leq s - p \leq 1 \) (Since right heavy Huffman code is used)

Reverse also holds.

**Independent Source: Independence Unknown to Crypt analyst**

In this subsection we assume that the four-symbol source is independent, but that the crypt analyst does not know this a priori.

**Construction of the Cross-Product-Machine to Determine the Probability Distribution on the Parser**

The Markov chains for the generator and the parser can be used to construct the **Cross-Product-Machine** (CPM). The CPM consists of pair states: \( \phi \alpha, \phi \beta, \phi \gamma, \theta \alpha, \theta \beta, \theta \gamma, \eta \alpha, \eta \beta, \) and \( \eta \gamma \).

Our CPM will have a 0-arrow from one pair-state, \( \phi \alpha \), to another, \( \theta \beta \), if:

1. In the generator machine, there is a 0-arrow from \( \phi \) to \( \theta \), and

2. In the parser machine, there is a 0-arrow from \( \alpha \) to \( \beta \).

Similarly, our CPM will have a 1-arrow from one pair state, \( \phi \alpha \), to another, \( \eta \alpha \) if:

1. In the generator machine, there is a 1-arrow from \( \eta \) to \( \eta \), and

2. In the parser machine, there is a 1-arrow from \( \alpha \) to \( \alpha \).

Let us choose the Huffman tree in Figure A.4 as the generator and create the Markov chain.

![Huffman tree and its Markov chain](image)

Figure A.4: A Huffman tree and its Markov chain
Let us choose the Huffman tree in Figure A.5 as the parser and create the Markov chain. Given the generator and the parser, as above the Cross-Product-Machine can be used to determine a probability distribution \((a, b, c, d)\), on the parser as follows.

We tabulate the transition probabilities in the following table.

<table>
<thead>
<tr>
<th></th>
<th>(\phi\alpha)</th>
<th>(\phi\beta)</th>
<th>(\phi\gamma)</th>
<th>(\theta\alpha)</th>
<th>(\theta\beta)</th>
<th>(\theta\gamma)</th>
<th>(\eta\alpha)</th>
<th>(\eta\beta)</th>
<th>(\eta\gamma)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\phi\alpha)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(p+q)</td>
<td>0</td>
<td>(r+s)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(\Phi\beta)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(p+q)</td>
<td>0</td>
<td>(r+s)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(\Phi\gamma)</td>
<td>0</td>
<td>0</td>
<td>(p+q)</td>
<td>0</td>
<td>0</td>
<td>(r+s)</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(\theta\alpha)</td>
<td>(q/p + q)</td>
<td>(p/P + q)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(r+s)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(\theta\beta)</td>
<td>(q/p + q)</td>
<td>0</td>
<td>(p/p + q)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(\theta\gamma)</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(\eta\alpha)</td>
<td>(s/r + s)</td>
<td>(r/r + s)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(\eta\beta)</td>
<td>(s/r + s)</td>
<td>0</td>
<td>(r/r + s)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(\eta\gamma)</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table A.4: Transition probabilities in the CPM or Transition matrix

\[\begin{bmatrix} A & B & C & D & E & F & G & H & I \end{bmatrix} \times \text{TransitionMatrix} = \begin{bmatrix} A & B & C & D & E & F & G & H & I \end{bmatrix}, \text{Where } P(\phi\alpha) = A, P(\phi\beta) = B, \ldots, P(\eta\gamma) = I\]

\[D(q/p + q) + E(q/p + q) + F + G(s/r + s) + H(s/r + s) + I = A\]
\[D(p/p + q) + G(r/r + s) = B\]
\[E(p/p + q) + H(r/r + s) = C\]
\[C(p + q) = D\]
\[A(p + q) = E\]
\[B(p + q) = F\]
\[A(r + s) + B(r + s) + C(r + s) = G\]
\[H = 0\]
\[I = 0\]
\[A + B + C + D + E + F + G + H + I = 1\]

Solving the above equations results in the stationary probabilities in terms of \(p, q, r\) and \(s\).

\[P(\phi \alpha) = A = -(1/2)(r - 1)/(1 + p + p^2)\]
\[P(\phi \beta) = B = (1/2)(r + rp + p^2)/(1 + p + p^2)\]
\[P(\Phi \gamma) = C = -(1/2)(p + 1)/p/(1 + p + p^2)\]
\[P(\theta \alpha) = D = -(1/2)(p + q)(r - 1)/p/(1 + p + p^2)\]
\[P(\theta \beta) = E = -(1/2)(p + q)(r - 1)/(1 + p + p^2)\]
\[P(\theta \gamma) = F = (1/2)(p + q)(r + rp + p^2)/(1 + p + p^2)\]
\[P(\eta \alpha) = G = (1/2)(r + s)\]
\[P(\eta \beta) = H = 0\]
\[P(\eta \gamma) = I = 0\]

We compute the probability distribution for the parser \((a, b, c, d)\) using the stationary probabilities above.

\[a = P[\gamma \rightarrow \alpha]/\text{const.}\]
\[= P[\phi \gamma \rightarrow \theta \alpha]P[\phi \gamma] + P[\theta \gamma \rightarrow \theta \alpha]P[\theta \gamma] + P[\eta \gamma \rightarrow \phi \alpha]P[\eta \gamma]/\text{const.}\]
\[= (p + q).(-1/2)(r - 1)p/(1 + p + p^2) + 1.1/2(p + q)(r + rp + p^2)/(1 + p + p^2)
\[+ 1.0/\text{const.}\]
\[= (p + q).(-1/2)(r - 1)p/(1 + p + p^2) + (1/2)(p + q)(r + rp + p^2)/(1 + p + p^2)/\text{const.}\]

\[b = P[\gamma \rightarrow \alpha]/\text{const.}\]
\[= P[\phi \gamma \rightarrow \eta \alpha]P[\phi \gamma] + P[\theta \gamma \rightarrow \phi \alpha]P[\theta \gamma] + P[\eta \gamma \rightarrow \Phi \alpha]P[\eta \gamma]/\text{const.}\]
\[= (r + s).(-1/2)(r - 1)p/(1 + p + p^2) + 1.1/2(p + q)(r + rp + p^2)/(1 + p + p^2)
\[+ 1.0/\text{const.}\]
\[= (r + s).(-1/2)(r - 1)p/(1 + p + p^2) + (1/2)(p + q)(r + rp + p^2)/(1 + p + p^2)/\text{const.}\]

\[c = P[\beta \rightarrow \alpha]/\text{const.}\]
\[= P[\phi \beta \rightarrow \eta \alpha]P[\phi \beta] + P[\theta \beta \rightarrow \phi \alpha]P[\theta \beta] + P[\eta \beta \rightarrow \phi \alpha]P[\eta \beta]/\text{const.}\]
\[= (r + s).(1/2)(r + rp + p^2)/(1 + p + p^2)
\[+ (q/p + q).- (1/2)(p + q)(r - 1)/(1 + p + p^2) + (s/r + s).0/\text{const.}\]
\[= (r + s).(1/2)(r + rp + p^2)/(1 + p + p^2) + (q).- (1/2)(r - 1)/(1 + p + p^2)/\text{const.}\]
\[ d = P[\alpha \rightarrow \alpha] / \text{const.} \]
\[ = P[\phi \alpha \rightarrow \eta \alpha]P[\phi \alpha] + P[\theta \alpha \rightarrow \phi \alpha]P[\theta \alpha] + P[\eta \alpha \rightarrow \phi \alpha]P[\eta \alpha] / \text{const.} \]
\[ = (r + s) - (1/2)(r - 1)/(1 + p + p^2) \]
\[ + (q/p + q) - (1/2)(p + q)(r - 1)p/(1 + p + p^2) + (r/r + s)(1/2)(r + s) / \text{const.} \]
\[ = (r + s) - (1/2)(r - 1)/(1 + p + p^2) + (q) - (1/2)(r - 1)p/(1 + p + p^2) \]
\[ + (r)(1/2) / \text{const.} \]

\[ \text{Const.} = [p^3 + (2r + q + 1)p^2 + (3r + q)p + (r + qr + 1)] / [2(1 + p + p^2)] \]

Substituting for \text{const.} and simplifying we get the values for a, b, c, and d as follows:

\[ a = [(p + q)(p + r + p^2)] / [p^3 + (2r + q + 1)p^2 + (3r + q)p + (r + qr + 1)] \]

\[ b = [- (r + s)(r - 1)p + (p + q)(r + rp + p^2)] / [p^3 + (2r + q + 1)p^2] \]
\[ + (3r + q)p + (r + qr + 1)] \]

\[ c = [(r + s)(r + rp + p^2) - (r - 1)(1 + p + p^2)] / [p^3 + (2r + q + 1)p^2] \]
\[ + (3r + q)p + (r + qr + 1)] \]

\[ d = [(r - 1)(- r - s - pq) + r(1 + p + p^2)] / [p^3 + (2r + q + 1)p^2 + (3r + q)p] \]
\[ + (r + qr + 1)] \]

**Theorem 1** (The right-heavy Huffman codes) Suppose a source independently produces letters P, Q, R, and S with respective probabilities p, q, r, and s. Suppose further that the stream of letters is encoded using the code word set \( C_1 = 00, 01, 10, 11 \) where \( s \geq r, q \geq p, (r + s) \geq (p + q)(r + s) \geq 1/2 \) and \( s \geq 1/4 \) then the encoding rule is ambiguous if and only if

\[
(p + q) \geq (r + s) \geq 1/2
\]
\[
(p - q)(r - 1) \geq (2(p + q) - (r + s))(r + rp + p^2)
\]
\[
(r - 1)(- r - s - pq + p(r + s) + (1 + p + 2p^2)) \geq (p + q)(p + r + p^2)
\]
\[
(r + s)(r + rp + p^2) - (r - 1)(1 + p + p^2) \geq (p + q)(p + r + p^2)
\]
\[
(r + s)(r + rp + p^2) - (r - 1)(1 + p + p^2) \geq -(r + s)(r - 1)p + (p + q)(r + rp + p^2)
\]
\[
(r - 1)(- r - s - pq) + r(1 + p + p^2)((r + s)(r + rp + p^2) - (r - 1)(1 + p + p^2)
\]

**Proof:** Assume that the encoding rule is ambiguous, i.e., the code word set \( C_0 = \)
000, 001, 01, 1 with respective probabilities \(a, b, c,\) and \(d\) is also right-heavy Huffman-consistent. For \(C_0\) to be right-heavy Huffman-consistent, its probability distribution must satisfy the following conditions:

\[
\begin{align*}
  d & \geq a + b + c \\
  c & \geq a + b \\
  b & \geq a \\
  c & \geq a \\
  c & \geq b \\
  d & \geq c
\end{align*}
\]

Replacing \(a, b, c,\) and \(d\) with the solutions above, results in

\[
\begin{align*}
  (1) & \Rightarrow (p + q) \geq (r + s) \geq 1/2 \\
  (2) & \Rightarrow (p - q)(r - 1) \geq (2(p + q) - (r + s))(r + rp + p2) \\
  (3) & \Rightarrow (r - 1)(-r - s - pq + p(r + s) + (1 + p + 2p2)) \geq (p + q)(p + r + p2) \\
  (4) & \Rightarrow (r + s)(r + rp + p2) - (r - 1)(1 + p + p2) \geq (p + q)(p + r + p2) \\
  (5) & \Rightarrow (r + s)(r + rp + p2) - (r - 1)(1 + p + p2) \geq -(r + s)(r - 1)p \\
  & \quad + (p + q)(r + rp + p2) \\
  (6) & \Rightarrow (r - 1)(-r - s - pq) + r(1 + p + p2) \geq (r + s)(r + rp + p2) \\
  & \quad - (r - 1)(1 + p + p2)
\end{align*}
\]

The above derivation is also reversible.

A.4.3 Conclusion

We have partially extended the cryptanalysis developed for Huffman coding with three symbol alphabet to four symbol alphabet. Here we have considered only a pair of Huffman trees to our analysis. This could be extended to all the other pairs for a complete analysis. Although this extension fails to yield an effective algorithm to break the code, it shows how difficult it is to cryptanalyse Huffman code with four symbols.
Bibliography


[27] Steven Pigeon, and Yoshua Bengio - A Memory Efficient Adaptive Coding Algorithm for Very Large Set of Symbols- Universite de Montreal, Repport technique #1081 and #1095


[29] Mojdeh Mohtashemi, On the Cryptanalysis of Huffman Codes, Master of Science thesis submitted at the Department of Computer Science, MIT.


[53] http://www.dai.ed.ac.uk/HIPR2/canny.htm#1


