Secure Wavelet-Based Coding of Images, and Application to Privacy Protected Video Surveillance

by

Karl Martin

A thesis submitted in conformity with the requirements for the degree of Doctor of Philosophy
Graduate Department of The Edward S. Rogers Sr. Dept. of Electrical and Computer Engineering
University of Toronto

Copyright © 2010 by Karl Martin
Abstract

Secure Wavelet-Based Coding of Images, and Application to Privacy Protected Video Surveillance

Karl Martin
Doctor of Philosophy
Graduate Department of The Edward S. Rogers Sr. Dept. of Electrical and Computer Engineering
University of Toronto
2010

The protection of digital images and video from unauthorized access is important for a number of applications, including privacy protection in video surveillance and digital rights management for consumer applications. However, traditional cryptographic methods are not well suited to digital visual content. Applying standard encryption approaches to the entire content can require significant computational resources due to the large size of the data. Furthermore, digital images and video often need to be manipulated, such as by resizing or transcoding, which traditional encryption would hinder. A number of image and video-specific encryption approaches have been proposed in the literature, but many of the them have significant negative impact on the ability to compress the data, which is a necessary requirement of most imaging systems.

In this work, a secure image coder, called Secure Set Partitioning in Hierarchical Trees (SecSPIHT), is proposed. It combines wavelet-based image coding (compression) with efficient encryption. The encryption is applied to a small number of selected bits in the code domain, to achieve complete confidentiality of all the content while having no negative impact on compression performance. The output of the system is a secure code that cannot be decrypted and decoded without the provision of a secret key. It has superior rate-distortion performance compared to JPEG and JPEG2000, and the bit-rate
can be easily scaled via a simple truncation operation. The computational overhead of the encryption operation is very low, typically requiring less than 1% of the coded image data to be encrypted.

A related secure object-based coding approach is also presented. Called Secure Shape and Texture Set Partitioning in Hierarchical Trees (SecST-SPIHT), it codes and encrypts arbitrarily-shaped visual objects. A privacy protection system for video surveillance is proposed, using SecST-SPIHT to protect private data, such as face and body images appearing in surveillance footage. During normal operation of the system, the private data objects are protected via SecST-SPIHT. If an incident occurs that requires access to the data (e.g., for investigation), a designated authority must release the key. This is superior to other methods of privacy protection which irreversibly blur or mask the private data.
Dedication

This work is dedicated to my fiancée, Cindy.

I am forever grateful for her support.
Acknowledgements

I give my thanks to my committee members, Profs. A. N. Venetsanopoulos, D. Hatzinakos, A. Khisti, and T. S. Abdelrahman. They provided many useful insights that allowed me to refine and improve the work presented in this thesis.

I would like to thank my supervisor, Prof. Kostas Plataniotis for providing me with valuable guidance throughout my career as a graduate student. His insistence on producing the highest quality work is a lesson that has benefited me greatly.

I would like to thank my colleagues in the Multimedia Lab. I’ve seen many people come and go during my time, but it’s always been a fun and interesting bunch.

I would like to thank my family for their support and for tolerating my seemingly endless years in school.

Finally, my undying gratitude goes to my fiancée Cindy. If I am to be successful at anything, it will be because of her support.
Contents

List of Acronyms viii

List of Symbols xi

List of Tables xiii

List of Figures xiii

1 Introduction 1

1.1 Video Surveillance and Privacy 3

1.2 Challenges and Motivations 4

1.3 Contributions 5

1.4 Publications and Patents 7

1.5 Organization 8

2 Background and Prior Art 9

2.1 Image Coding 9

2.2 Wavelet-Based Image Coding 12

2.2.1 Colour Transform 14

2.2.2 Discrete Wavelet Transform 17

2.2.3 Embedded Coding 23

2.3 Coding of Arbitrarily-Shaped Visual Objects 31
## 2.3.1 Shape-Adaptive Discrete Wavelet Transform

2.3.2 Shape and Texture SPIHT Coding Algorithm

### 2.4 Image Encryption

#### 2.4.1 Pixel Domain Encryption

#### 2.4.2 Transform Domain Encryption

#### 2.4.3 Code Domain Encryption

### 2.5 Privacy Protection in Video Surveillance

### 2.6 Chapter Summary

### 3 Secure Image Coder

#### 3.1 Definitions and Methodology

#### 3.2 SecSPIHT Algorithm

#### 3.3 Security Analysis

##### 3.3.1 Attack I – Brute Force Attack on $B_e$

##### 3.3.2 Attack II – Progressive Attack on $B_e$

##### 3.3.3 Attack III – Attack on Unencrypted Code Portion

##### 3.3.4 Summary of Security Analysis

#### 3.4 Simulation Results

#### 3.5 Chapter Summary

### 4 Secure Visual Object Coder

#### 4.1 Definitions and Methodology

#### 4.2 SecST-SPIHT Algorithm

#### 4.3 System for Privacy Protection in Video Surveillance

#### 4.4 Simulation Results

#### 4.5 Chapter Summary

### 5 Conclusion

#### 5.1 Future Work
5.1.1 Addressing Error Control and Resiliency .......................... 116
5.1.2 Addressing Inter-Frame Prediction in Video (Motion Compensation) 117
5.1.3 Stream Cipher Selection and Setup ................................. 118
5.1.4 Alternative Cryptographic Approaches ................................. 118
5.1.5 Key Management Considerations .................................... 118
5.1.6 Extension to Other Coding Approaches ................................. 119

A Discrete Wavelet Transform Derivation and Properties 120
A.1 Orthogonal Wavelets ...................................................... 120
A.2 Biorthogonal Wavelets .................................................... 128

B Selected Wavelet Filters 131
B.1 Cohen-Daubechies-Feauveau 9/7 DWT (Floating Point) ............... 131
B.2 Le Gall 5/3 DWT (Integer) .............................................. 133

Bibliography 135
# List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>Advanced Encryption Standard</td>
</tr>
<tr>
<td>ASIC</td>
<td>Application-Specific Integrated Circuit</td>
</tr>
<tr>
<td>bpp</td>
<td>Bits per Pixel</td>
</tr>
<tr>
<td>CCTV</td>
<td>Closed Circuit Television</td>
</tr>
<tr>
<td>CDF</td>
<td>Cohen-Daubechies-Feauveau</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>DCT</td>
<td>Discrete Cosine Transform</td>
</tr>
<tr>
<td>DES</td>
<td>Data Encryption Standard</td>
</tr>
<tr>
<td>DRM</td>
<td>Digital Rights Management</td>
</tr>
<tr>
<td>DWT</td>
<td>Discrete Wavelet Transform</td>
</tr>
<tr>
<td>EBCOT</td>
<td>Embedded Block Coding with Optimal Truncation</td>
</tr>
<tr>
<td>EZW</td>
<td>Embedded Zero-Tree Wavelet</td>
</tr>
<tr>
<td>fps</td>
<td>Frames Per Second</td>
</tr>
<tr>
<td>HVS</td>
<td>Human Visual System</td>
</tr>
<tr>
<td>ICT</td>
<td>Irreversible Colour Transform</td>
</tr>
<tr>
<td>JPEG</td>
<td>Joint Pictures Experts Group</td>
</tr>
<tr>
<td>LIP</td>
<td>List of Insignificant Pixels</td>
</tr>
<tr>
<td>LIS</td>
<td>List of Insignificant Sets</td>
</tr>
<tr>
<td>LSB</td>
<td>Least Significant Bit</td>
</tr>
<tr>
<td>Acronym</td>
<td>Definition</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
</tr>
<tr>
<td>LSP</td>
<td>List of Significant Pixels</td>
</tr>
<tr>
<td>MPEG</td>
<td>Motion Pictures Experts Groups</td>
</tr>
<tr>
<td>MRA</td>
<td>Multiresolution Analysis</td>
</tr>
<tr>
<td>MSB</td>
<td>Most Significant Bit</td>
</tr>
<tr>
<td>MSE</td>
<td>Mean Squared Error</td>
</tr>
<tr>
<td>pdf</td>
<td>Probability Density Function</td>
</tr>
<tr>
<td>PSNR</td>
<td>Peak Signal to Noise Ratio</td>
</tr>
<tr>
<td>RCT</td>
<td>Reversible Colour Transform</td>
</tr>
<tr>
<td>ROI</td>
<td>Region-of-Interest</td>
</tr>
<tr>
<td>SA-DCT</td>
<td>Shape-Adaptive Discrete Cosine Transform</td>
</tr>
<tr>
<td>SA-DWT</td>
<td>Shape-Adaptive Discrete Wavelet Transform</td>
</tr>
<tr>
<td>SAQ</td>
<td>Successive Approximation Quantization</td>
</tr>
<tr>
<td>SCS</td>
<td>Shape Code Set</td>
</tr>
<tr>
<td>SecSPIHT</td>
<td>Secure Set Partitioning in Hierarchical Trees</td>
</tr>
<tr>
<td>SecST-SPIHT</td>
<td>Secure Shape and Texture Set Partitioning in Hierarchical Trees</td>
</tr>
<tr>
<td>SOT</td>
<td>Spatial Orientation Tree</td>
</tr>
<tr>
<td>SPIHT</td>
<td>Set Partitioning in Hierarchical Trees</td>
</tr>
<tr>
<td>ST-SPIHT</td>
<td>Shape and Texture Set Partitioning in Hierarchical Trees</td>
</tr>
<tr>
<td>VGA</td>
<td>Video Graphics Array</td>
</tr>
</tbody>
</table>
### List of Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathbb{R} )</td>
<td>Set of real numbers</td>
</tr>
<tr>
<td>( \mathbb{Z} )</td>
<td>Set of integer numbers</td>
</tr>
<tr>
<td>( \mathbb{N} )</td>
<td>Set of natural numbers</td>
</tr>
<tr>
<td>( \psi(t) )</td>
<td>Mother wavelet function</td>
</tr>
<tr>
<td>( \phi(t) )</td>
<td>Wavelet scaling function</td>
</tr>
<tr>
<td>( H, h )</td>
<td>DWT low-pass reconstruction filter and filter coefficients</td>
</tr>
<tr>
<td>( H', h' )</td>
<td>DWT low-pass decomposition filter and filter coefficients</td>
</tr>
<tr>
<td>( G, g )</td>
<td>DWT high-pass reconstruction filter and filter coefficients</td>
</tr>
<tr>
<td>( G', g' )</td>
<td>DWT high-pass decomposition filter and filter coefficients</td>
</tr>
<tr>
<td>( c_m )</td>
<td>Coarse (low-pass) DWT coefficients at scale ( m )</td>
</tr>
<tr>
<td>( d_m )</td>
<td>Detail (high-pass) DWT coefficients at scale ( m )</td>
</tr>
<tr>
<td>( \mathcal{T} )</td>
<td>Set of wavelet coefficients (SPIHT)</td>
</tr>
<tr>
<td>( \mathcal{O}(i,j) )</td>
<td>Offspring of coordinates ((i,j)) (SPIHT)</td>
</tr>
<tr>
<td>( \mathcal{D}(i,j) )</td>
<td>Descendants of coordinates ((i,j)) (SPIHT)</td>
</tr>
<tr>
<td>( \mathcal{L}(i,j) )</td>
<td>Descendants of coordinates ((i,j)), excluding offspring (SPIHT)</td>
</tr>
<tr>
<td>( S_n )</td>
<td>Significance test at bit-plane ( n ) (SPIHT)</td>
</tr>
<tr>
<td>( x(i,j) )</td>
<td>Pixel value at coordinates ((i,j))</td>
</tr>
<tr>
<td>( x_T(i,j) )</td>
<td>Transform coefficient value at coordinates ((i,j))</td>
</tr>
<tr>
<td>( s(i,j) )</td>
<td>Binary shape mask value at coordinates ((i,j))</td>
</tr>
<tr>
<td>( \mathcal{G} )</td>
<td>Set of all coordinates inside a visual object (ST-SPIHT)</td>
</tr>
<tr>
<td>( \alpha_p )</td>
<td>Alpha pixel test function (ST-SPIHT)</td>
</tr>
<tr>
<td>( \alpha_{SD} )</td>
<td>Alpha set-discard test function (ST-SPIHT)</td>
</tr>
<tr>
<td>( \alpha_{SR} )</td>
<td>Alpha set-retain test function (ST-SPIHT)</td>
</tr>
<tr>
<td>Symbol</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>Shape code level parameter (ST-SPIHT)</td>
</tr>
<tr>
<td>$B_n$</td>
<td>Set of code bits for bit-plane $n$ (SecSPIHT)</td>
</tr>
<tr>
<td>$B_{n,LIP}, B_{n,LIS}, B_{n,LSP}$</td>
<td>Code bits for LIP, LIS, and LSP coding phases (SecSPIHT)</td>
</tr>
<tr>
<td>$B_{n,LIP-sig}, B_{n,LIP-\text{sgn}}$</td>
<td>Significance and sign bits for LIP coding phase (SecSPIHT)</td>
</tr>
<tr>
<td>$B_{n,LIS-sig}, B_{n,LIS-Tsig}, B_{n,LIS-\text{sgn}}$</td>
<td>Significance and sign bits for LIS coding phase (SecSPIHT)</td>
</tr>
<tr>
<td>$B_{n,LIP-\alpha}, B_{n,LIS-\alpha}$</td>
<td>Alpha test bits for the LIP and LIS coding phases (SecST-SPIHT)</td>
</tr>
<tr>
<td>$S_{LIP-sig}, S_{LIP-\text{sgn}}$</td>
<td>LIP phase Markov states (SecSPIHT)</td>
</tr>
<tr>
<td>$S_{LIS-sig}, S_{LIS-Tsig}, S_{LIS-\text{sgn}}$</td>
<td>LIS phase Markov states (SecSPIHT)</td>
</tr>
<tr>
<td>$k_E, k_D$</td>
<td>Encryption and decryption keys</td>
</tr>
<tr>
<td>$K$</td>
<td>Security parameter (SecSPIHT)</td>
</tr>
<tr>
<td>$f_E$</td>
<td>Stream cipher encryption function (SecSPIHT)</td>
</tr>
<tr>
<td>$f_{cE}$</td>
<td>Controlled stream cipher encryption function (SecSPIHT)</td>
</tr>
<tr>
<td>$f_D$</td>
<td>Stream cipher decryption function (SecSPIHT)</td>
</tr>
<tr>
<td>$f_{cD}$</td>
<td>Controlled stream cipher decryption function (SecSPIHT)</td>
</tr>
<tr>
<td>$B_e$</td>
<td>Set of encrypted bits (SecSPIHT)</td>
</tr>
</tbody>
</table>
## List of Tables

2.1 Example: reconstruction of decimal value 115 using binary successive approximation quantization (bit-plane coding) ........................................ 25

2.2 Advantages and disadvantages of encryption in different domains. ........ 44

3.1 SPIHT bit classification .......................................................... 62

3.2 “Lena” – Number of bits in each classification for first 6 coding iterations 80

3.3 “Map” – Number of bits in each classification for first 6 coding iterations 81

3.4 “Document” – Number of bits in each classification for first 6 coding iterations ................................................................. 81

3.5 “Tulip” – Number of bits in each classification for first 6 coding iterations 82

3.6 The number of bits encrypted for the test images using different values of 
    $K$. The percentage of bits encrypted for the test images coded at 1 bpp 
    is shown in brackets. ............................................................. 82

4.1 ST-SPIHT bit classification ....................................................... 88

4.2 Percentage of frame occupied by test objects. .............................. 101

B.1 CDF 9/7 filter coefficients approximated to 10 decimal places [1] ....... 132
## List of Figures

2.1 Generic wavelet-based colour image coding and decoding system . . . . 13  
2.2 Example input image “Lena” ........................................ 14  
2.3 RGB representation of example input image “Lena” .................. 15  
2.4 YC\textsubscript{b}C\textsubscript{r} representation of example input image “Lena” calculated using the ICT ........................................ 16  
2.5 Two-level DWT decomposition and reconstruction .................. 18  
2.6 Two-dimensional, two-level DWT decomposition ..................... 19  
2.7 Two-dimensional, two-level DWT decomposition of “Lena”, luminance (Y) channel ........................................ 19  
2.8 DWT lifting implementation ............................................ 20  
2.9 Parent-child structure of wavelet coefficients in the SPIHT algorithm . 27  
2.10 SPIHT coder ............................................................. 30  
2.11 Example visual object .................................................... 32  
2.12 Shape and Texture SPIHT (ST-SPIHT) coding system .......... 34  
2.13 One-level 2-D SA-DWT using in-place lifting DWT implementation . 34  
2.14 Interleaved subband arrangement ..................................... 35  
2.15 ST-SPIHT Coder ......................................................... 37  
2.16 Shape Code Set (SCS) Subroutine ................................. 38  
2.17 Generic image encryption system ................................. 39  

xiv
2.18 “Lena” test image and encrypted version, before and after SPIHT coding. The pre-compression, pixel domain encryption significantly reduces coding performance. .......................................................... 46

2.19 PSNR vs. bit-rate for “Lena” image and encrypted version using random pixel permutation, coded using SPIHT ................................. 47

2.20 Reconstruction of only partial DCT coefficients. First column contains low frequency components 1 to $n$ (to be encrypted); second column contains high frequency components $n+1$ to 64 (to not be encrypted); third column contains contrast enhancement of column 2. First row: $n = 10$; second row: $n = 32$; third row: $n = 50$. .............................................. 50

2.21 Reconstruction of “Lena” image based on the first level, high frequency DWT subbands (HL1, LH1, and HH1), with automatic contrast enhancement. .................................................. 52

2.22 Generalized privacy protection system for image and video input . . . 55

3.1 SPIHT bit-stream ................................................................. 61

3.2 Composition of subset $B_n$ of SPIHT bit-stream ............................. 61

3.3 Graph of Markov process during SPIHT LIP coding/decoding phase . . 63

3.4 Graph of Markov process during SPIHT LIS coding/decoding phase . . 64

3.5 SecSPIHT secure image coding and decoding system .......................... 66

3.6 SecSPIHT secure image coder .................................................. 67

3.7 SecSPIHT secure image decoder ............................................... 68

3.8 SecSPIHT coder algorithm ....................................................... 69

3.9 SecSPIHT test output (at 1.0 bpp); $K = 2$; 990 bits encrypted (0.38% of total code) ................................................................. 71

3.10 Reconstruction from first coding iteration (highest bit-plane) ............. 75

3.11 “Lena” image and encrypted version (decrypted with incorrect key) ... 78

3.12 “Map” image and encrypted version (decrypted with incorrect key) ... 79
3.13 “Document” image and encrypted version (decrypted with incorrect key) 79
3.14 “Tulip” image and encrypted version (decrypted with incorrect key) . . . 80
3.15 Coding performance of the SecSPIHT secure coder, JPEG2000 and JPEG 83

4.1 Composition of subset $B_n$ of ST-SPIHT bit-stream .................. 88
4.2 SecST-SPIHT secure object-based coding and decoding system ........ 89
4.3 SecST-SPIHT secure object coder ........................................ 91
4.4 SecST-SPIHT secure object decoder ....................................... 91
4.5 SecST-SPIHT coder algorithm ............................................. 92
4.6 Secure Shape Code Set (SCS) Subroutine ............................. 93
4.7 Generic video surveillance system ........................................ 95
4.8 Proposed system for privacy protection in video surveillance ......... 97
4.9 Example surveillance frames with privacy protection measures ........ 100
4.10 ‘Surveillance1’ test object .................................................. 101
4.11 ‘Surveillance2’ test object .................................................. 102
4.12 ‘Akiyo’ test object ......................................................... 102
4.13 ‘Foreman’ test object ....................................................... 102
4.14 ‘Surveillance1’ test object/frame decoded and decrypted output ($K = 2$): (a)/(d) with correct key; (b)/(e) with incorrect key; (c)/(f) with incorrect key and shape provided externally ............................................. 105
4.15 ‘Surveillance1’-rect test object/frame decoded and decrypted output ($K = 2$): (a)/(d) with correct key; (b)/(e) with incorrect key; (c)/(f) with incorrect key and shape provided externally ............................................. 106
4.16 ‘Surveillance2’ test object/frame decoded and decrypted output ($K = 2$): (a)/(d) with correct key; (b)/(e) with incorrect key; (c)/(f) with incorrect key and shape provided externally ............................................. 107
4.17 ‘Surveillance2’-rect test object/frame decoded and decrypted output \( (K = 2) \): (a)/(d) with correct key; (b)/(e) with incorrect key; (c)/(f) with incorrect key and shape provided externally .......................... 108

4.18 ‘Akiyo’ test object/frame decoded and decrypted output \( (K = 2) \): (a)/(d) with correct key; (b) with incorrect key; (c)/(e) with incorrect key and shape provided externally .......................... 109

4.19 ‘Akiyo’-rect test object/frame decoded and decrypted output \( (K = 2) \): (a)/(d) with correct key; (b) with incorrect key; (c)/(e) with incorrect key and shape provided externally .......................... 110

4.20 ‘Foreman’ test object/frame decoded and decrypted output \( (K = 2) \): (a)/(d) with correct key; (b) with incorrect key; (c)/(e) with incorrect key and shape provided externally .......................... 111

4.21 ‘Foreman’-rect test object/frame decoded and decrypted output \( (K = 2) \): (a)/(d) with correct key; (b) with incorrect key; (c)/(e) with incorrect key and shape provided externally .......................... 112

4.22 The fraction of bits encrypted vs. the security level parameter \( K \) (number of encrypted coding iterations) for different \( \lambda \) (shape code levels). The total bits in the code corresponds to a bit-rate of 2.4 bits-per-object-pixel. 114

A.1 Daubechies 4-tap wavelet ................................. 121
Chapter 1

Introduction

We live in an increasingly networked world. The previous generation of computing was dominated by isolated, desktop applications. However, today, we see the rise in popularity of online services. Many of these are considered to be “social networking,” such as Facebook, YouTube, Flickr, etc. Much of these interactions involves the generation and exchange of media content — specifically, images and video. Fueled by increasing bandwidth and more innovative tools, user generated content is growing fast to dominate the Internet [2].

We see a similar rise of the use of rich media content in enterprise environments. Workers are increasingly “telecommuting,” where co-worker interactions occur via video conference services, such as Skype. Additionally, with the rise in consumer demand of rich media services, many business have consequently focussed on this sector as their primary business interest.

However, with all these advances, there are serious questions related to security and privacy. Services will typically have protection measures in place to control access to the service. However, increased connectivity means that it is usually easy for content to leak outside of the control of the service. For example, a Facebook user might post a photo which is only accessible to a select number of individuals. But if one user then copies the
Chapter 1. Introduction

content to a public website, the image will be propagated without any controls. In this case, the original user that posted the photo has lost complete control over the content.

Clearly, in order to address these issues, protection controls must be tied directly to the media content. However, the direct protection of rich media is still a difficult task. A variety of cryptographic tools are available to make content confidential, to digitally sign it, to ensure integrity, etc. However, the sheer volume of data associated with images and video will often make the use of these tools cumbersome or infeasible. Additionally, the unique way in which image and video data is distributed and consumed (e.g., we want to resize it, transcode it, etc.) limits the applicability of these tools.

The work in this thesis addresses the problem of how to keep visual data confidential. This specific area has many applications, such as in digital rights management (DRM) and privacy protection. “Whole content” encryption of images and video often requires too many computational resources to be practical. Furthermore, it hinders the intelligent manipulation of the content, such as bit-rate and resolution scaling. A variety of research works have focussed on “selective” encryption approaches for images and video [3]. However, the methods are often ad-hoc, with little in terms of security analysis. Many (arguably, the majority) of the schemes proposed have been found to be insecure [4].

In this thesis, an image encryption approach, called Secure Set Partitioning in Hierarchical Trees (SecSPIHT) is proposed. Based on the well-known, and highly efficient, SPIHT coding algorithm [5], the proposed scheme is a secure coder which combines selective encryption with efficient, embedded wavelet-based coding. While the general approach bears some similarity to other works [3], the design methodology is formalized and rigorous. Unlike other methods, a comprehensive security analysis is provided. It is shown that, with appropriate parameter selection, complete confidentiality of digital colour images can be achieved with the scheme, requiring encryption of typically less than 1% of the coded data. It also retains all the desirable properties of SPIHT, such as
superior rate-distortion performance and a fully embedded bit-stream, supporting fine-grained rate-distortion scalability. When SecSPIHT is applied to an image, a secret key is used to maintain confidentiality. Users cannot reveal any aspect of the visual content if they do not have the key.

1.1 Video Surveillance and Privacy

One of the primary applications of interest addressed in this thesis is privacy protection in video surveillance. Video surveillance of public and private spaces is ubiquitous and indiscriminate [6]. There is increasing concern over the abuse of video surveillance content through voyeurism and other illegitimate uses [7]. Video surveillance indiscriminately captures a plethora of private data, such as facial images (revealing identity) and the activities and behaviours of subjects in the footage. In most cases, the subjects are not involved in unauthorized activities, so there is no legitimate use for this private data. The organizations that utilize video surveillance systems will usually have legitimate reasons for deploying them (e.g., to maintain physical security) and have no desire to infringe on anyone’s privacy. However, there is a distinct lack of technical tools which allow organizations to protect content that contains private visual data of the public. Since the data must be accessed for regular usage, basic access controls provide little protection in terms of prevention of abuse.

One of the primary technical challenges is that, in surveillance video, the private content (e.g., facial images) is interspersed with the non-private data. Hence, it is the separation of the private data from the non-private data that is key to designing privacy protection measures.

In this thesis, a secure object-based coder, SecST-SPIHT, is proposed as a tool for privacy protection in video surveillance. It codes and encrypts arbitrarily shaped objects which are segmented from the background. Using detection and tracking technology (e.g.,
for face), the private regions can be identified and then protected as an isolated object. This separation and protection of the private data offers effective privacy protection, while not impeding the functionality of the surveillance system. For example, by defining facial images as the private data, these can be removed and protected, while the surveillance personnel can still monitor the general activities within the footage. If an incident occurs that requires revealing the identity of a subject, an authorized person can release a key that is used to decrypt the private objects. The encryption is completely reversible, hence there is no loss of functionality in terms of investigative use of the footage.

This approach can be contrasted with other methods which only perform blurring or masking as a post-processing operation [8]. In these cases, the unblurred originals must be retained in case an incident requires investigation. This is clearly a weaker form of privacy protection, since the original can still fall into unauthorized hands.

Additionally, the object-based approach to privacy protection offers additional flexibility not found in other systems. Specifically, since objects are coded individually, the associated coding and encryption parameters, such as bit-rate, key, security level, etc., can be controlled on an individual object level.

1.2 Challenges and Motivations

There are a number of technical challenges that have motivated the work presented in this thesis. They are summarized as follows:

- Efficient encryption of visual data would be an enabling tool for many applications, but it is still an open research question. Clearly, it is desirable to have cryptographic tools designed specifically for images and video that would support the unique functionality of this data type. One of the primary challenges is how to make the entire content confidential while minimizing computational overhead due to the large volume of data.
In addressing the first technical challenge above, one is motivated to explore “selective” encryption approaches which only encrypt a small portion of the data. However, this gives rise to the next logical challenge: “what should we encrypt?” While this question has been addressed by several researchers, the approach is often ad-hoc. Therefore, the work in this thesis attempts to take a much more formalized and rigorous approach. Additionally, the interaction between the encryption approach and compression is often neglected. Since compression is a necessary part of almost any image or video system, the encryption approach should always be evaluated in this context.

Given the largely ad-hoc approach to the design of selective encryption methods, the next fundamental challenge is to provide a more comprehensive security analysis of proposed methods than what currently exists. This is often difficult because the potential attacks suffered by such schemes are usually not the same types of traditional attacks considered in cryptography. Assessing the security of an image or video selective encryption scheme requires detailed knowledge of the data type as well as the unique features of the encryption approach.

In the area of privacy protection for video surveillance, the primary challenge to be considered is how to protect private visual data that may be an arbitrary region within non-private data. The application requires that non-private data remain available, so the task becomes how to appropriately separate the two and protect the private content.

### 1.3 Contributions

The contributions of the work presented in this thesis are summarized as follows.

**Domain-based Analysis of Generalized Encryption Approaches (Section 2.4):**
In the Background and Prior Art chapter, the general advantages and disadvantages of performing selective encryption in different domains is covered. The three domains considered are: i) pixel (spatial) domain; ii) transform domain (DCT or DWT); and iii) code domain. This type of detailed analysis is currently missing in the literature.

**Secure Image Coder — SecSPIHT (Chapter 3):** The Secure Set Partitioning in Hierarchical Trees (SecSPIHT) secure image coder is proposed. SecSPIHT is used to simultaneously code (compress) and encrypt digital colour images. This is a code-domain selective encryption approach, applied to SPIHT [5]. Complete confidentiality is achieved while requiring encryption of less than 1% of the coded data. The image reveals no visual details unless the correct decryption key is provided. The approach is developed in a rigorous manner, accompanied by a comprehensive security analysis. SecSPIHT retains all the desirable features of SPIHT, including superior rate-distortion performance and a fully embedded bit-stream, supporting fine-grained rate-distortion scalability.

**Secure Visual Object Coder — SecST-SPIHT (Chapter 4):** The Secure Shape and Texture SPIHT (SecST-SPIHT) secure visual object coder is proposed. It codes and encrypts the shape and texture of arbitrarily-shaped visual objects. As with SecSPIHT, it is a code-domain selective encryption approach applied to the visual object coder ST-SPIHT [9, 10] (the prior work of this author). Complete confidentiality of the object is achieved while requiring the encryption of a small portion of the code bits. SecST-SPIHT may be set to protect just the object texture or both the object texture and shape. It retains the desirable properties of ST-SPIHT. This includes having a single coding routine that codes both the shape and texture simultaneously, producing a single, easily managed bit-stream.

**System for Privacy Protection in Video Surveillance (Section 4.3):** A system for
privacy protection in video surveillance is proposed, utilizing SecST-SPIHT to secure private object data. The system separates private objects (such as the face) to be coded and encrypted using SecST-SPIHT. The operation of the surveillance system is not impeded by the privacy measures; personnel can monitor general activities without invading privacy. If an incident occurs and legitimate investigation of some of the private objects is required, an authorized agent releases a key to provide access to the data. By encrypting the objects soon after capture, maximum privacy protection is achieved.

1.4 Publications and Patents

The work presented in this thesis has been published in the following journal and conference papers:

**Secure Image Coder (SecSPIHT):**


**Secure Visual Object Coder (SecST-SPIHT):**


Chapter 1. Introduction

Note that the SecST-SPIHT relies heavily on this author’s prior work on ST-SPIHT, published in:


The following patent has been filed for SecST-SPIHT:


1.5 Organization

The remainder of this thesis is organized as follows. Chapter 2 covers the background and prior art in the areas of wavelet-based image coding, coding of arbitrarily-shaped objects, image encryption, and privacy protection in video surveillance. The reader is referred to Appendix A for a more detailed discussion on the derivation of the discrete wavelet transform. Appendix B describes the selected wavelet filters utilized in the proposed schemes. Chapter 3 presents the proposed SecSPIHT secure image coder, including the detailed design methodology and security analysis. Simulation results are provided at the end of the chapter. Chapter 4 presents the proposed SecST-SPIHT secure visual object-based coder as well as a system for privacy protection in video surveillance. Simulation results are included at the end of this chapter. The thesis concludes with Chapter 5.
Chapter 2

Background and Prior Art

This chapter covers the background material and prior art relevant to the proposed secure coding techniques. The sections “Image Coding” (2.1) and “Wavelet-Based Image Coding” (2.2) discuss fundamental coding techniques upon which the proposed secure coder is built. The section “Coding of Arbitrarily-Shaped Objects” (2.3) discusses the shape-adaptive coding techniques upon which the proposed secure object-based coder is built. The sections “Image Encryption” (2.4) and “Privacy Protection in Video Surveillance” (2.5) discuss the prior art in those respective areas. The chapter concludes with a summary in Section 2.6.

2.1 Image Coding

Image coding refers to the coding of digital images to create a new representation to serve certain application requirements. Most commonly, image coding is used for compression — i.e., the coding system produces a representation (code) that is shorter (in bits) than the original image.¹ The growth and proliferation of digital imaging and video applications can be seen to be symbiotically related to advances in compression technology.

¹According to convention, the term “image coding” implies “image compression.” This terminology is used throughout this thesis.
Efficient image and video compression is a key enabling technology for modern, graphics rich applications, and the growth of such applications provides strong motivation for continuous advancements in compression technology.

To illustrate the necessity for digital image compression in many applications, consider a typical colour image of size $512 \times 512$ pixels. Such an image occupies $1/5^{th}$ the screen area of a modern computer monitor of size $1280 \times 1204$. Full colour images are typically represented by 24 bits-per-pixel — 3 colour channels (e.g., red, green, and blue), with 8 bits per colour channel. Uncompressed, the example image requires 768 KB of storage space. In graphics rich applications, where many images must be stored and transmitted over bandwidth-limited channels such as consumer broadband Internet connections, this volume of image data may represent an insurmountable obstacle to the usability of such applications if compression technologies are not available.

In general, we consider $M \times N$ input digital colour images of the form

$$\mathbf{x} : \mathbb{Z}^2 \rightarrow \mathbb{Z}^3,$$  \hspace{1cm} (2.1)

representing a two-dimensional matrix of three-component RGB (red, green, blue) colour samples

$$\mathbf{x}(i, j) = [x(i, j)_1, x(i, j)_2, x(i, j)_3],$$ \hspace{1cm} (2.2)

with $i = 0, 1, ..., M-1$ and $j = 0, 1, ..., N-1$ denoting the spatial position of the pixel, and $x(i, j)_k$ denoting the component in the red ($k = 1$), green ($k = 2$), or blue ($k = 3$) colour channel. The pixels have $B$-bit precision (also known as “bit-depth”), with integer values in the range $[0, 2^B - 1]$. There are other possible digital colour image representations, both integer and non-integer, however, the aforementioned representation is most commonly used in the design of image coding systems since it is the common representation used in computer graphics systems.

The output of an image coding system is a binary string, representing the coded image. Image coding schemes are classified as either: i) lossless; or ii) lossy. Lossless image
coding schemes produce representations that, when decoded, are bit-for-bit identical with the original image. Conversely, lossy coding produces output that, when decoded, will differ from the original image. These schemes usually employ some form of quantization to reduce the image data, hence producing quantization distortion or noise. However, the output from a lossy image coding scheme may be classified as “visually” or “perceptually” lossless in cases where the distortion in the decoded image is imperceptible to a typical human observer. Lossy image coding is prevalent in most applications involving photographic-style, “natural” images, since the compression achieved is usually much greater than what can be achieved with lossless image coding. Additionally, lossy coding schemes typically offer an operating parameter which can be used to control the code size (i.e., achieved compression) vs. output distortion.

The performance of an image coding system is usually evaluated via two measures: i) bit-rate; and ii) quality. For an $M \times N$ image, the bit-rate (in units bits-per-pixel — bpp) and is calculated as follows:

$$\text{Bit-Rate (bpp)} = \frac{\text{Total code size (bits)}}{MN}$$  \hspace{1cm} (2.3)

Alternatively, compression ratio [11] may be used, calculated as follows:

$$\text{Compression Ratio} = \frac{\text{Total code size (bits)}}{\text{Uncompressed image size (bits)}} \hspace{1cm} (2.4)$$

$$= \frac{\text{Compressed Bit-Rate (bpp)}}{\text{Uncompressed Bit-Rate (bpp)}} \hspace{1cm} (2.5)$$

The image quality is typically measured by comparing the decoded image with the original uncompressed image [12]. Some image quality measures (or, conversely, image distortion measures) may be purely numerical in nature, measuring the error in the decoded image. In cases where the decoded image is intended for human consumption, some quality measures take into account the properties of the human visual system (HVS) to determine “perceptual quality” or “perceptual distortion” [13, 14]. The most common measures, Mean Squared Error (MSE) and Peak Signal-to-Noise Ratio (PSNR), do not
take into account the HVS, and are calculated as follows:

\[
MSE = \frac{1}{3MN} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} \sum_{k=1}^{3} (y(i, j)_k - x(i, j)_k)^2,
\]

(2.6)

\[
PSNR = 10 \log_{10} \left( \frac{(2^B - 1)^2}{MSE} \right),
\]

(2.7)

where \( x \) and \( y \), are the original and decoded images, respectively.

The performance of image coding schemes is compared by comparing their quality vs. bit-rate curves. Depending on the intended application, coding schemes may be compared at a specific bit-rate to determine which produces better quality output. Alternatively, the schemes may be compared at a particular quality level to determine which requires a lower bit-rate.

### 2.2 Wavelet-Based Image Coding

The most popular, modern image coding schemes are the class of so-called transform-based coders. These image coders use transforms—typically frequency decompositions—to provide decorrelation and energy compaction in the spatial domain, before quantization is applied. This reduces spatial redundancy as well as providing a representation that closely relates to HVS models, thus supporting intelligent control over distortion when quantization is applied. After quantization, the transform coefficients will typically be coded using approaches such as run length encoding and/or entropy coding, to produce the final binary output.

The discrete cosine transform (DCT) has traditionally been the most common transform utilized in transform-based image coding schemes. The DCT is used in the ubiquitous JPEG standard [15]. However, more recently, the discrete wavelet transform (DWT) has gained interest, with wavelet-based image coding systems showing greater performance compared to DCT-based systems. The DWT is utilized in the newer JPEG2000 standard [16].
A generic wavelet-based image coding and decoding system is shown in Figure 2.1. Each system component will be discussed in the following sections. It should be noted that the class of so-called “embedded” wavelet-based image coders (discussed in Section 2.2.3) perform quantization implicitly during coefficient coding. Since the image encryption approach proposed in this thesis is built upon this class of coders, the system diagram in Figure 2.1 does not show quantization and coding as a separate system blocks.

Additionally, it should be noted that while the work in this thesis focuses on colour images, the described systems will generally accept monochrome (greyscale) input images, requiring only minor modifications. Specifically, for monochrome input, the colour transform can be removed from the system, and all subsequent processing blocks applied to only a single channel of data.
2.2.1 Colour Transform

The input images are assumed to be represented in an RGB colour space. For natural images, the RGB colour representation is highly correlated [17]. This is demonstrated in the example image shown in Figure 2.2, with RGB channels shown in Figure 2.3. It is clearly evident that the RGB channels are highly correlated, with energy largely dispersed across the channels. Since the purpose of the image coding system is to produce a reduced representation of the image data, a colour transform is utilized to decorrelate the colour channels and to compact most of the perceptual information content into a single channel (i.e., “energy compaction”). The transform is performed on each colour vector individually (i.e., does not take into account spatial relationships) and outputs a the vector in the new colour space.

The field of colour science and colour space representations is broad and largely beyond the scope of this work. The discussion that follows will focus on the $Y_{C_b}C_r$ (luminance, chrominance blue, chrominance red) representation, which is ubiquitous in image coding systems since it provides a high level of decorrelation for natural imagery.

We consider two transforms, both defined in the JPEG2000 standard [16, 18]. The first transform, the so-called Irreversible Colour Transform (ICT), maps the integer RGB input values to the traditional floating point $Y_{C_b}C_r$ representation. As the name implies, the ICT may not be exactly reversible due to limited precision representation of the
Chapter 2. Background and Prior Art

(a) R Channel  (b) G Channel  (c) B Channel

Figure 2.3: RGB representation of example input image “Lena”

YC\textsubscript{b}C\textsubscript{r} values, and is intended for lossy image coding systems where the loss induced by quantization or coding is far greater than what occurs during reverse of the ICT. The second transform, the so-called Reversible Colour Transform (RCT), maps the integer RGB input values to integer values that approximate the YC\textsubscript{b}C\textsubscript{r} representation. This transform is exactly reversible and is intended for use in lossless image coding systems.

Utilizing RGB input images in the form described in Equations 2.1 and 2.2 (substituting \( k = [1, 2, 3] \) with \([R, G, B]\), respectively, for clarity), the ICT is defined as follows [16, 18]:

\[
x(i, j)_{Y} \triangleq \alpha_R x(i, j)_R + \alpha_G x(i, j)_G + \alpha_B x(i, j)_B \quad (2.8)
\]
\[
x(i, j)_{C_b} \triangleq \frac{0.5}{1 - \alpha_B} (x(i, j)_B - x(i, j)_Y) \quad (2.9)
\]
\[
x(i, j)_{C_r} \triangleq \frac{0.5}{1 - \alpha_R} (x(i, j)_R - x(i, j)_Y) \quad (2.10)
\]

where

\[
\alpha_R \triangleq 0.299, \quad \alpha_G \triangleq 0.587, \quad \alpha_B \triangleq 0.114
\]

Equations 2.8 to 2.10 define the transform; it can be approximated with the following
Chapter 2. Background and Prior Art

Figure 2.4: YCbCr representation of example input image “Lena” calculated using the ICT matrix equation:

\[
\begin{pmatrix}
  x(i,j)_Y \\
  x(i,j)_{Cb} \\
  x(i,j)_{Cr}
\end{pmatrix} =
\begin{pmatrix}
  0.299 & 0.587 & 0.114 \\
  -0.169 & -0.331 & 0.500 \\
  0.500 & -0.419 & -0.081
\end{pmatrix}
\begin{pmatrix}
  x(i,j)_R \\
  x(i,j)_G \\
  x(i,j)_B
\end{pmatrix}
\]

(2.11)

The YCbCr representation of the example image is shown in Figure 2.4. As can be seen, the channels have been largely decorrelated (compared to the RGB representation, shown in Figure 2.3) with most of the energy compacted into the Y channel.

The reverse ICT can be derived trivially by reversing Equations 2.8 to 2.10.

The RCT is limited to integer arithmetic and is defined as follows [16, 18]:

\[
x(i,j)_Y' \equiv \left\lfloor \frac{x(i,j)_R + 2x(i,j)_G + x(i,j)_B}{4} \right\rfloor
\]

(2.12)

\[
x(i,j)_{Db} \equiv x(i,j)_B - x(i,j)_G
\]

(2.13)

\[
x(i,j)_{Dr} \equiv x(i,j)_R - x(i,j)_G
\]

(2.14)

The RCT can be exactly inverted using the following relationships:

\[
x(i,j)_G = x(i,j)_Y' - \left\lfloor \frac{x(i,j)_{Db} + x(i,j)_{Dr}}{4} \right\rfloor
\]

(2.15)

\[
x(i,j)_B = x(i,j)_{Db} + x(i,j)_G
\]

(2.16)

\[
x(i,j)_R = x(i,j)_{Dr} + x(i,j)_G
\]

(2.17)
Chapter 2. Background and Prior Art

It should be noted that given the RGB input with $B$-bit precision, the luminance channel $Y'$ will also have $B$-bit precision, whereas the $D_b$ and $D_r$ channels experience an increase in dynamic range such that they require $(B + 1)$-bit precision.

As will be discussed in Section 2.2.3 (Embedded Coding), the class of embedded wavelet-based coders can operate in both lossy and lossless modes if a pure integer processing pipeline is utilized. This is accomplished by the use of the RCT and an integer-to-integer wavelet transform (discussed in the next section). However, if, for a given application, it is known that lossless coding is not required, the floating-point pipeline is usually preferred since the ICT (and the subsequent floating-point wavelet transform) provide better energy compaction, resulting in better overall system performance in terms of quality vs. bit-rate. However, the integer-only processing operations may be selected in cases where computational efficiency is of primary interest.

### 2.2.2 Discrete Wavelet Transform

The purpose of the transform in transform-based image coding schemes is to provide decorrelation and energy compaction in the spatial domain. Traditionally, the discrete cosine transform (DCT) has been the transform of choice, being utilized in the JPEG standard [15] and the MPEG video coding standards [19, 20, 21]. However, more recently, there has been interest in utilizing the discrete wavelet transform (DWT) as it has been shown to offer greater decorrelation and energy compaction when applied to natural imagery.

The DWT performs a subband decomposition where the bases are naturally localized in space. Hence, it provides a simultaneous space-frequency representation, within the limits of the Heisenberg Uncertainty Principle [22]. This has the desirable side-effect of negating the need for performing the transform on blocks of data, as is required for the DCT (e.g., the $8 \times 8$ blocks used in the JPEG standard). In practice, the DWT may be applied to a signal of arbitrary size without tiling. This has the benefit of avoiding
Chapter 2. Background and Prior Art

The “blocking” artifacts which are characteristic of DCT-based coding schemes when operating at low bit-rates.

A detailed discussion on the derivation and properties of the DWT is provided in Appendix A. In summary, a particular DWT is usually defined via specification of the low-pass filter coefficients \( h(n) \) and the high-pass filter coefficients \( g(n) \). The filters \( h(-n) \) and \( g(-n) \) form low-pass \( (H') \) and high-pass \( (G') \) decomposition filters, respectively. A one-level discrete wavelet transform (DWT) involves filtering an input discrete time or space signal, \( x(n) \), using \( H' \) and \( G' \), and then downsampling by a factor of two to create the subbands \( x_{L1}(n) \) and \( x_{H1}(n) \), respectively. The next level DWT subbands, \( x_{L2}(n) \) and \( x_{H2}(n) \), can be obtained by repeating the procedure using \( x_{L1}(n) \) as the new input. The entire system of wavelet decomposition (analysis) and reconstruction (synthesis) is shown in Figure 2.5. An arbitrary number of levels of decomposition may be performed up to the point where a subband signal length is equal to 1. Typically the number of levels is chosen to achieve a maximum degree of energy compaction.

The reverse transform involves upsampling the subband signals, filtering with the corresponding low-pass filter, \( h(n) \) \( (H) \), or high-pass filter, \( g(n) \) \( (G) \), and element-wise addition.

A two-dimensional DWT can be performed by applying a one-dimensional DWT separably on the rows and columns of the input [23]. The subbands are named: LL
Chapter 2. Background and Prior Art

Figure 2.6: Two-dimensional, two-level DWT decomposition

Figure 2.7: Two-dimensional, two-level DWT decomposition of “Lena”, luminance (Y) channel

(low-pass on rows, low-pass on columns), LH (low-pass on rows, high-pass on columns), HL (high-pass on rows, low-pass on columns), and HH (high-pass on rows, high-pass on columns) as illustrated in Figure 2.6. To perform the second level DWT, the LL subband acts as the new input and the process is repeated. Typically 4 to 6 levels are performed in image coding schemes applied to screen sized images. A two-dimensional, two-level DWT decomposition of an image is shown in Figure 2.7. It can be seen that most of the energy of the original image is stored in the LL subband. This makes the DWT an effective transform for image coding.
Chapter 2. Background and Prior Art

Lifting Implementation

The lifting scheme [24] is an alternative DWT implementation which offers the following advantages over the traditional, convolutional based implementation [25, 26]:

- Fast implementation. By making optimal use of similarities between the high and low-pass filters, the number of floating point operations can be reduced by a factor of two.

- The lifting scheme can be performed in-place, replacing the original pixel values with the DWT coefficients, without additional extra memory.

- Simple invertibility. The steps for the forward transform are simply reversed to perform the inverse transform.

Any FIR wavelet filter may be factorized into simpler lifting filters, $\lambda_i$, to be used in the lifting implementation of the DWT [18].

The basic lifting structure is shown in Figure 2.8. The procedure involves first separating the input sequence, $x(n)$, into even ($x(2n) = y_0^{(0)}(n)$) and odd ($x(2n + 1) = y_1^{(0)}(n)$) subsequences. This is often called the lazy wavelet transform. Each subsequent step of the lifting scheme involves alternating between updating each of the subsequences with a filtered version of the other.

![Figure 2.8: DWT lifting implementation](image)
For each step \( l \), the lifting filter \( \lambda_l \) is applied to either the odd or even subsequence. For odd \( l \), the even subsequence is filtered and added to the odd subsequence:

\[
y_1^{(l)}(n) = y_1^{(l-1)}(n) + \sum_i \lambda_l(i) y_0^{(l-1)}(n - i).
\] (2.18)

For even \( l \), the odd subsequence is filtered and added to the even subsequence:

\[
y_0^{(l)}(n) = y_0^{(l-1)}(n) + \sum_i \lambda_l(i) y_1^{(l-1)}(n - i).
\] (2.19)

After the last lifting step, the even and odd subsequences are weighted with factors \( K_0 \) and \( K_1 \), respectively. The final odd subsequence \( (y(2n)) \) is the low-pass output and the even subsequence \( (y(2n+1)) \) is the high-pass output. If the lifting operation is performed in-place, the output is the single interleaved subband sequence \( y(n) \).

The benefits of the lifting scheme are greatly enhanced for the case of odd length symmetric FIR wavelet filters, with high and low-pass filters differing in length by 2 (e.g., the selected CDF 9/7 and Le Gall 5/7 biorthogonal wavelet filters — see Appendix B) [18]. In this case, the lifting filters \( \lambda_l \) are simply 2-tap symmetric. For odd \( l \), Equation 2.18 is reduced to:

\[
y_1^{(l)}(n) = y_1^{(l-1)}(n) + \alpha_l \left( y_0^{(l-1)}(n) + y_0^{(l-1)}(n + 1) \right),
\] (2.20)

with \( \alpha_l \) being the scalar value of the two equal coefficients of \( \lambda_l \). For even \( l \), the lifting step is:

\[
y_0^{(l)}(n) = y_0^{(l-1)}(n) + \alpha_l \left( y_1^{(l-1)}(n) + y_1^{(l-1)}(n + 1) \right).
\] (2.21)

The specific filter coefficients and implementation steps for the selected CDF 9/7 and Le Gall 5/3 wavelet filters are provided in Appendix B.

**Integer Implementations**

The clear invertibility that the lifting implementation offers allows for the introduction of non-linear perturbations into the forward transform, without loss of invertibility [18].
One of the most beneficial non-linearities is the ability to modify wavelet transforms in order to map integers to integers to be used for lossless image coding [27].

The lifting step shown in Equation 2.18 is modified as follows [27, 18]:

$$y_1^{(l)} = y_1^{(l-1)}(n) + \left[ \frac{1}{2} + \sum_i \lambda_l(i) y_0^{(l-1)}(n-i) \right].$$  \hfill (2.22)

By applying the same modification for even $l$ and skipping the final weighting factors $K_0$ and $K_1$, the result is a wavelet transform which outputs integer coefficients. The integer DWT is easily inverted by reversing the lifting steps. Note that the skipping of the final weighting factors can be accounted for during quantization and coding in order to maintain appropriate normalization [18].

For the special case of odd length symmetric FIR wavelet filters that differ in length by 2 (Equations 2.20 and 2.21), $\lambda_l$ is 2-tap symmetric so the integer variant becomes the following simple operations:

$$y_1^{(l)} = y_1^{(l-1)}(n) + \left[ \frac{1}{2} + \alpha_l \left( y_0^{(l-1)}(n) + y_0^{(l-1)}(n+1) \right) \right],$$  \hfill (2.23)

and

$$y_0^{(l)} = y_0^{(l-1)}(n) + \left[ \frac{1}{2} + \alpha_l \left( y_1^{(l-1)}(n) + y_1^{(l-1)}(n+1) \right) \right].$$  \hfill (2.24)

Again, $\alpha_l$ is the scalar value of the two equal coefficients of $\lambda_l$. In special cases, $\alpha_l$ is a power of 2, giving

$$\alpha_l = (-1)^s \frac{2^p}{2^q},$$

where $p$ and $q$ are positive integers and $s$ is either 0 or 1 depending on the sign of $\alpha_l$. Equation 2.23 becomes

$$y_1^{(l)} = y_1^{(l-1)}(n) + \left[ \frac{1}{2} + (-1)^s \frac{2^p}{2^q} \left( y_0^{(l-1)}(n) + y_0^{(l-1)}(n+1) \right) \right] \quad \text{and} \quad y_0^{(l)} = y_0^{(l-1)}(n) + \left[ \frac{1}{2} + (-1)^s \frac{2^p}{2^q} \left( y_1^{(l-1)}(n) + y_1^{(l-1)}(n+1) \right) \right].$$

A similar result is achieved for Equation 2.24. Since the input terms are integers, Equation 2.25 involves only simple bit shifts, additions and/or subtractions, and truncation. This makes the transform very computationally efficient.
Chapter 2. Background and Prior Art

The Le Gall 5/3 integer DWT uses this framework. The details are provided in Appendix B.2.

2.2.3 Embedded Coding

There are a variety of approaches to quantization and coding of wavelet coefficients [28, 5, 18]. Arguably the most popular general approach is the so-called “embedded coding” methodology. Embedded coders are an application of successive refinement techniques [29], whereby the code is produced in a progressive manner such that, for a given input, the low bit-rate code (low quality) is the prefix of higher bit-rate codes (higher quality). Formally, given code streams $S_1$ and $S_2$ from the same image, with lengths $N_1$ and $N_2$ respectively, if $N_1 < N_2$, then the first $N_1$ bits of $S_2$ equals $S_1$. The shorter sequence $S_1$ is said to be embedded in the longer sequence $S_2$. The code may be progressively decoded from low quality to high quality, with the decoding operation terminated at an arbitrary point to achieve a certain desired bit-rate or quality. This is known as successive approximation quantization (SAQ). In summary, embedded coders have the following desirable properties:

- Bit-rate may be set to an arbitrary value
- A single coded image can be repurposed to lower bit-rates via simple truncation (i.e., simple rate-distortion scalability)
- If the wavelet coefficients are integers, the coding may be applied progressively from lossy reconstruction to lossless reconstruction

The first comprehensive embedded wavelet-based image coder was the Embedded Zerotree Wavelet (EZW) coder [28]. This achieved remarkable performance, which was then further improved with the Set Partitioning in Hierachical Trees (SPIHT) coder [5]. The JPEG2000 standard uses a form of embedded coding on coefficient blocks, known
as Embedded Block Coding with Optimal Truncation (EBCOT) [16, 18]. This supports multiple modes of scalability, such as resolution and spatial (image cropping), which is achieved via a post-compression optimization routine which orders and combines block code portions according to the scalability criteria. Compared to EZW and SPIHT, this comes at the cost of slightly reduced compression performance.

All of the wavelet-based embedded image coders use a binary form of SAQ which can be viewed as a form of bit-plane coding. The next section describes the concepts behind bit-plane coding. The section after that describes the SPIHT coding approach which is utilized as the platform for the secure image coder proposed in this thesis.

**Bit-Plane Coding**

Bit-plane coding is essentially the binary case of SAQ. Specifically, it is an iterative quantization method wherein the number of quantizer reconstruction points doubles at each iteration and they are uniformly distributed. In practice, signal values are first represented by their most significant bit (MSB) — a 1-bit quantizer — then the lower significant bits are added one-by-one at each iteration. The reconstruction (approximation) values are selected to be at the mid-point between the quantizer partitions (i.e., the source is assumed to have uniform pdf).

This is best illustrated through an example. Table 2.1 shows the bit-plane coding of the integer value 115. In practice, the value may be floating point or integer. With integer values, perfect reconstruction is achieved when bit-plane 0 is coded. With floating point values, there is generally no point at which perfect reconstruction is guaranteed. Coding is usually terminated once a certain bit-rate or quality (distortion) criterion is met.

It should be noted that with each subsequent iteration, the set of quantizer reconstruction points does not include the set from the previous iteration [9]. The implication of this that the reconstruction error is not guaranteed to monotonically decrease with
Chapter 2. Background and Prior Art

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Bit-plane</th>
<th>Known Bits</th>
<th>Approximation</th>
<th>Decimal Approximation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>1xxxxxx</td>
<td>1100000</td>
<td>96</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>11xxxxx</td>
<td>1110000</td>
<td>112</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>111xxxxx</td>
<td>1111000</td>
<td>120</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1110xxx</td>
<td>1110100</td>
<td>116</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>11100xx</td>
<td>1110010</td>
<td>114</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>111001x</td>
<td>1110011</td>
<td>115</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1110011</td>
<td>1110011</td>
<td>115</td>
</tr>
</tbody>
</table>

Table 2.1: Example: reconstruction of decimal value 115 using binary successive approximation quantization (bit-plane coding)

Each iteration. This can be seen in the example, going from iteration 2 to iteration 3. However, in practice, over a large number of coefficients, the total distortion will generally decrease with each iteration.

Despite the name, bit-plane coding itself is in fact a method of quantization, not coding. In practice, after bit-plane coding, the bit-planes must be coded using some form of run-length or entropy coding. The EZW and SPIHT coding approaches utilize trees of coefficients with similar statistical properties to produce efficient codes.

Set Partitioning in Hierarchical Trees

The Set Partitioning in Hierarchical Trees (SPIHT) [5] embedded wavelet-based image coding scheme is a generalization and improvement of the Embedded Zerotree Wavelet (EZW) coder [28]. It directly outputs a binary stream that does not require further entropy coding. The output code may be arbitrarily truncated to achieve a desired bit-rate or quality, with bit-level accuracy.

The input to the coder is $x_T$, the DWT of the image of the image $x$. The following is a high-level summary of the coding steps (a detailed description is provided afterward):

1. Organize wavelet coefficients into spatial orientation trees (SOT) with low-frequency
coefficients at the top and high frequency at the bottom

2. Set initial bit-plane \( n = n_{\text{max}} \) based on MSB of highest magnitude coefficient; initialize a significance threshold \( 2^n \)

3. Bit-plane code trees; after each iteration: \( n \leftarrow n - 1 \); each iteration has two passes:

   (a) Sorting Pass:
   
   - Check whether trees have any significant coefficients (\( \geq 2^n \)):
     - If no: tree is considered a “zerotree” and no further action is required for this tree during this iteration
     - If yes: tree is partitioned into new trees and significance test repeated on smaller trees and individual coefficients released at the top of the trees. Partitioning and testing is repeated until all remaining trees are zerotrees. Coefficients found significant are stored in a list
   - The result of each significance test (yes/no) is output as a bit in the output codestream. For each coefficient found significant, its sign is output as well.

   (b) Refinement Pass:

   - Output \( n^{\text{th}} \) bit in each coefficient found significant at previous iterations

4. Terminate coding when bit-rate or quality criterion is met

The spatial orientation tree (SOT) structure utilized in SPIHT is shown in Figure 2.9. The coefficients in the LL subband are grouped into \( 2 \times 2 \) blocks. The top-left coefficient of each block is not the parent of any coefficients and does not join any tree. The top-right coefficient is the parent of the four coefficients in the adjacent HL subband located in the same position as the original \( 2 \times 2 \) block in the LL subband. The bottom-right is the parent of the four corresponding coefficients in the adjacent HH subband, and so on.
Each of these first level children is parent to four coefficients corresponding to the same spatial position, in the same subband direction (HL, LH, or HH), but one level lower. Aside from the root (in the LL subband) each tree will only contain either HL, LH, or HH coefficients (not a mixture). If the LL subband is $M \times N$ in size, the algorithm will initialize with $3M\times N/4$ trees, since 3 out of 4 LL subband coefficients is the root of a tree. A tree or a branch of a tree may be referred to as a set of pixels.$^2$

The implication of the SOT structure is that a single tree contains coefficients corresponding to a compact spatial region in the original image, containing all the HL, LH, or HH coefficients derived from that region. This is intended to exploit a feature called “self-similarity” across subbands: spatially localized image features will likely be present in the corresponding location in many of the subbands. The pyramid structure, with lower frequency components at the top and higher frequency components at the bottom, exploits the fact that much of the signal energy will typically be compacted in the lower frequency subbands (top of the tree).

$^2$Said and Pearlman used the term “pixels” when referring to the coefficients in a wavelet decomposition [5].
The SPIHT coding algorithm uses some formalized definitions with respect to a given node at location \((i,j)\) in an arbitrary subband:

- \(\mathcal{O}(i,j)\): set of all offspring coordinates of node \((i,j)\) (one level below in the spatial orientation tree);
- \(\mathcal{D}(i,j)\): set of all descendant coordinates of node \((i,j)\) (all levels below in the spatial orientation tree);
- \(\mathcal{H}\): set of coordinates of all coefficients in LL subband;
- \(\mathcal{L}(i,j) = \mathcal{D}(i,j) - \mathcal{O}(i,j)\). That is, all the descendants of location \((i,j)\), but not including the direct offspring.

Key to the algorithm is the following significance function:

\[
S_n(\mathcal{T}) = \begin{cases} 
1, & \max_{(i,j) \in \mathcal{T}} \{|x_{\mathcal{T}}(i,j)|\} \geq 2^n, \\
0, & \text{otherwise},
\end{cases} 
\]  

(2.26)

where \(\mathcal{T}\) is a set of coordinates, and \(x_{\mathcal{T}}(i,j)\) is the particular coefficient at location \((i,j)\). The set \(\mathcal{T}\) is said to be significant with respect to quantization level \(n\) if \(S_n(\mathcal{T}) = 1\). The set is said to be insignificant otherwise. If a single pixel is being tested for significance, \(S_n(\{(i,j)\})\) is written as \(S_n(i,j)\) for simplified notation.

The algorithm employs three ordered lists: the list of insignificant sets (LIS), the list of insignificant pixels (LIP), and the list of significant pixels (LSP). The entries in the lists are denoted by the coordinates \((i,j)\). The entries in the LIP and LSP represent single pixels, while the entries in the LIS represent either \(\mathcal{D}(i,j)\) (labelled type “A”) or \(\mathcal{L}(i,j)\) (labelled type “B”).

For each quantization level \(n\), SPIHT has two passes. In the sorting pass, the LIP is evaluated for significant pixels which are then sent to the LSP. As well, the LIS is evaluated for sets with significant coefficients, which are partitioned until you have again
insignificant sets (zerotrees). The isolated significant pixels are sent to the LSP and the isolated insignificant pixels are sent to the LIP.

The refinement pass scans the LSP and transmits the next precision bit, corresponding to bit-plane \( n \), of each element in the list.

The encoder sends 0’s and 1’s depending on the decisions made during partitioning and the bits from the refinement pass. There is no requirement for entropy coding of a symbol set.

The complete algorithm is shown in Figure 2.10. The algorithm is terminated when the desired maximum bit-rate or quality is achieved.

A subtle but important result of the algorithm is that when type A sets in the LIS are found significant, partitioned, and reinserted as type B entries at the end of the list, these type B entries are tested for significance during the same quantization pass. This forms the partitioning procedure which isolates significant pixels and attempts to form large insignificant sets at each quantization level.

The coding efficiency of the algorithm is achieved by exploiting the self-similarity across subbands. That is, if the coefficients at the root of a tree are insignificant, it is likely that the descendants are insignificant as well. In this case, only a single bit is required to code the entire current bit-plane of all the coefficients in the tree (this is the zerotree case). This feature is due to the fact that a single SOT corresponds to a compact spatial region in the original image, with the lower frequency subbands at the top of the tree. If the tree does contain significant coefficients, they are likely to be near the top of the tree, requiring minimal tree partitioning.

To decode an image coded using SPIHT, the same algorithm is followed except at each binary decision point, instead of outputting a bit, the next sequential bit from the code stream is taken as input, and the appropriate reconstruction decision made based on the value. This makes the codec symmetric with the coding and decoding execution path being identical. For this to work, the decoder must simply be initialized with the
Input: $x_T$

1. **Initialization:** Find initial quantization level $n = \left\lceil \log_2 \left( \max_{i,j} \{|x_T(i,j)|\} \right) \right\rceil$; set LSP as an empty list, and add all $(i, j) \in \mathcal{H}$ to the LIP and those with descendants to the LIS as type A entries.

2. **Sorting pass:**
   
   2.1. For each entry $(i, j)$ in the LIP:
      
      2.1.1. Output $S_n(i, j)$;
      
      2.1.2. If $S_n(i, j) = 1$ then move $(i, j)$ to the LSP and output the sign of $x_T(i, j)$;
   
   2.2. For each entry $(i, j)$ in the LIS:
      
      2.2.1. If type A entry:
         
         • Output $S_n(\mathcal{D}(i, j))$;
         
         • If $S_n(\mathcal{D}(i, j)) = 1$ then:
            
            − For each $(k, l) \in \mathcal{O}(i, j)$:
               
               * Output $S_n(k, l)$;
               
               * If $S_n(k, l) = 1$ then add $(k, l)$ to the LSP and output sign of $x_T(k, l)$;
               
               * If $S_n(k, l) = 0$ then add $(k, l)$ to the LIP;
            
            − If $\mathcal{L}(i, j) \neq \emptyset$ then move $(i, j)$ to the end of the LIS as type B entry; else, remove $(i, j)$ from the LIS;
      
      2.2.2. If type B entry:
         
         • Output $S_n(\mathcal{L}(i, j))$;
         
         • If $S_n(\mathcal{L}(i, j)) = 1$ then:
            
            − Add each $(k, l) \in \mathcal{O}(i, j)$ to the end of the LIS as type A entry;
            
            − Remove $(i, j)$ from the LIS.

3. **Refinement pass:** For each entry in the LSP, except those found significant in the current sorting pass, output the $n^{th}$ most significant bit of $|x_T(i, j)|$;

4. **Quantization-step update:** Decrement $n$ by 1 and go to Step 2.

Figure 2.10: SPIHT coder
original image size and the number of DWT levels. With this information, the decoder can generate all the SOTs (with empty coefficient values — this is the information that will be decoded) and initialize the LIS and LIP in the same way that the coder does. The decoder then starts reading in the codestream bits and replicates the decisions made by the coder. Decoding may terminate at any time to achieve an exact desired bitrate or image quality.

The decoder must also update the approximated coefficient values based on the decisions made. When a coefficient is first found to be significant at quantization level \( n \) and inserted into the LSP, it is known that \( 2^n \leq |x_T(i, j)| < 2^{n+1} \). Therefore, the coefficient is approximated as \( \hat{x}_T(i, j) = \pm 1.5 \times 2^n \) (the sign is chosen as per the explicitly transmitted sign bit). During the refinement pass of quantization level \( n \), the coefficient approximation is updated by adding or subtracting \( 2^{n-1} \) depending on whether the bit was 1 or 0, respectively.

Note that, for colour images, a slightly modified SOT structure can be employed to link trees across colour channels [30]. This has been called Color-SPIHT or CSPIHT.

### 2.3 Coding of Arbitrarily-Shaped Visual Objects

Arbitrarily-shaped visual objects (simply “visual objects,” for short) are images that are not necessarily rectangular in shape. A visual object is defined by its shape and texture (the actual pixel values). An example visual object is shown in Figure 2.11. Visual objects will typically have some semantic meaning. For example, a face, body, car, or some other physical object. Many applications can make use of image data represented as objects, such as indexing and retrieval, object recognition, and scene compositing.

The object shape is defined by an \( M \times N \) binary shape mask, in the following form:

\[
s : \mathbb{Z}^2 \rightarrow \{0, 1\},
\]

representing a two-dimensional matrix of binary values with \( i = 0, 1, ..., M - 1 \) and
Chapter 2. Background and Prior Art

32

(a) Object  (b) Object Shape Mask  (c) Object Texture

Figure 2.11: Example visual object

\( j = 0, 1, ..., N - 1 \) denoting the spatial position and where \( s(i, j) = 1 \) denotes spatial positions “inside” the object, and \( s(i, j) = 0 \) denotes spatial positions “outside” the object.

The texture \( \mathbf{x} \) is a full colour image, defined in the same way as a traditional image (Equation 2.1), except that spatial positions outside the object (as defined by \( s \)) have undefined values which are ignored in practice.

There are a variety of methods for coding of visual objects (known as object-based coding or shape-adaptive coding) [31, 32, 33, 34, 35, 36, 37, 38, 39]. The MPEG4 standard includes an object-based coding method, based on the shape-adaptive DCT (SA-DCT) [31] for video, and the shape-adaptive DWT (SA-DWT) [34] for still frames [21].

It is beyond the scope of this thesis to cover all the object-based coding methods; a comprehensive discussion can be found in [9]. The discussion here will focus on Shape and Texture SPIHT (ST-SPIHT) [9, 10] which is the selected object-based coding method utilized for the secure object-based coder proposed in this thesis. ST-SPIHT is the prior work of this author.

As with all object-based coding methods, it is necessary to code both the binary shape mask and the texture. However, the ST-SPIHT method has the following unique features:
• Single coding scheme to code both the shape and the texture, producing a single, combined embedded code

• Does not require first decoding the shape before decoding of the texture

• Fine rate-distortion scalability of both the shape and the texture

• Parameter to control the relative progression of the shape and texture portions of the embedded code

The ST-SPIHT coding system is shown in Figure 2.12. The following is a high-level summary of the coding scheme [9, 10] (a detailed description is provided afterward):

• Modified, in-place lifting SA-DWT is applied to the texture

• SOTs are created as per SPIHT, with individual coefficients being labelled as either “inside” or “outside” the object based on the shape mask

• SPIHT coding routine is modified to incorporate new binary tests to determine whether whole trees and individual coefficients are inside or outside the object; the results of these binary tests are output to the codestream.

The following sections provide details on the core components of the ST-SPIHT coding system — namely, the modified SA-DWT and the actual ST-SPIHT coding algorithm.

2.3.1 Shape-Adaptive Discrete Wavelet Transform

The ST-SPIHT object-based coder utilizes a variation of the Shape-Adaptive Discrete Wavelet Transform (SA-DWT), first introduced in [34]. The one-level SA-DWT is illustrated in Figure 2.13.

The operation involves performing an *in-place* lifting DWT on individual contiguous segments, first in the row direction, then in the column direction (or vice versa). The result is in an interleaved subband arrangement, as illustrated in Figure 2.14.
Chapter 2. Background and Prior Art

Figure 2.12: Shape and Texture SPIHT (ST-SPIHT) coding system

Figure 2.13: One-level 2-D SA-DWT using in-place lifting DWT implementation
An important feature of this approach to the SA-DWT is that the original shape representation is maintained in the transform domain, resulting an unambiguous, reversible transform domain representation of the object (shape and texture) which may be coded.

Multiple transform levels are performed in-place, applied only to the previous level LL subband coefficients, as with the traditional dyadic DWT.

### 2.3.2 Shape and Texture SPIHT Coding Algorithm

The ST-SPIHT coding algorithm uses the same fundamental constructs as the SPIHT coder. The input is the SA-DWT of the texture, $x_T$ and the shape mask $s$. The SOTs are the same as in SPIHT, created from $x_T$, however, in this case of arbitrarily-shaped visual objects, some coefficients will be outside the object and will have undefined values which should be ignored. The ST-SPIHT coder introduces a number of tests to determine whether coefficients are inside the object. It codes the results of those tests so that the shape can be reconstructed by the decoder. The SOT structure is used to code large blocks of the shape efficiently.
The set of all coordinates inside the object is defined as follows:

\[ G = \{(i, j) \mid s(i, j) = 1\}. \] (2.28)

The set of all coordinates outside the object is defined as follows:

\[ \overline{G} = \{(i, j) \mid s(i, j) = 0\}. \] (2.29)

Therefore, \( G \cup \overline{G} = \{(i, j) \mid i = 0, 1, \ldots, M-1, j = 0, 1, \ldots, N-1\} \) and \(|G| + |\overline{G}| = MN\).

A series of three “α-test” functions are introduced. These are the “shape” counterparts to the significance tests in SPIHT. The “α pixel test” function, \( \alpha_p(\cdot, \cdot) \), identifies whether a coordinate is inside or outside the shape and is defined follows:

\[ \alpha_p(i, j) = \begin{cases} 1, & (i, j) \in G \\ 0, & \text{otherwise} \end{cases}. \] (2.30)

The “α set-discard test” function, \( \alpha_{SD}(\cdot) \), identifies sets of coefficients that are entirely outside the object:

\[ \alpha_{SD}(\mathcal{T}) = \begin{cases} 0, & \mathcal{T} \subseteq \overline{G} \\ 1, & \text{otherwise} \end{cases}. \] (2.31)

where \( \mathcal{T} \) represents a given set of coefficients. Finally, the “α set-retain test” function, \( \alpha_{SR}(\cdot) \), identifies sets of coefficients that are entirely inside the object:

\[ \alpha_{SR}(\mathcal{T}) = \begin{cases} 1, & \mathcal{T} \subseteq G \\ 0, & \text{otherwise} \end{cases}. \] (2.32)

The ST-SPIHT coding algorithm requires a shape code level parameter, \( \lambda \), to be input. This defines the quantization level at which the routine forces the coding of not-yet-coded shape mask values \( s(i, j) \). This is done by applying the subroutine “Shape Code Set” (SCS) to the appropriate trees. The complete description of the ST-SPIHT routine and the SCS subroutine are shown in Figures 2.15 and 2.16, respectively. Note that, unlike SPIHT, colour coding is taken directly into account in the algorithm, with the inclusion
Input: $x_T$, $s$, $\lambda$

1. **Initialization**: Find initial quantization level $n = n_{\text{max}} = \left\lfloor \log_2 \left( \max_{(i,j,k)} \{|x_T(i,j)| \} \right) \right\rfloor$; set LSP = $\emptyset$; set LIP = $\mathcal{H}$; set LIS = $(i,j)_k$ “type-A” | $(i,j)_k \in \mathcal{H}$, $\mathcal{D}(i,j)_k \neq \emptyset$.

2. **Sorting pass**:
   2.1. For each $(i,j)_k \in \text{LIP}$:
      2.1.1. If $\alpha_p(i,j)$ not coded yet then output $\alpha_p(i,j)$;
      2.1.2. If $\alpha_p(i,j) = 1$ then:
         • Output $S_n(D(i,j)_k)$;
         • If $S_n(D(i,j)_k) = 1$ then move $(i,j)_k$ to the LSP and output the sign of $x_T(i,j)_k$;
      2.1.3. If $\alpha_p(i,j) = 0$ then remove $(i,j)_k$ from the LIP;
   2.2. For each entry $(i,j)_k \in \text{LIS}$:
      If “type-A” entry, $T = \mathcal{D}(i,j)_k$; If “type-B” entry, $T = \mathcal{L}(i,j)_k$
      2.2.1. If $n \geq \lambda$ and shape not completely coded, then:
         • If $\alpha_{SD}(T)$ not coded yet then output $\alpha_{SD}(T)$;
         • If $\alpha_{SD}(T) = 0$ then remove $(i,j)_k$ from the LIS and move on to next entry in the LIS (go to Step 2.2);
         • If $\alpha_{SD}(T) = 1$ then:
            – If $\alpha_{SR}(T)$ not coded yet then output $\alpha_{SR}(T)$;
            – If $\alpha_{SR}(T) = 0$ and $n = \lambda$ then run SCS($T$);
      2.2.2. If shape completely coded and $\alpha_{SD}(T) = 0$ then remove $(i,j)_k$ from the LIS and move on to next entry in the LIS (go to Step 2.2);
      2.2.3. If “type-A” entry and $\alpha_{SD}(T) = 1$:
         • Output $S_n(D(i,j)_k)$;
         • If $S_n(D(i,j)_k) = 1$ then:
            * For each $(p,q)_r \in \mathcal{O}(i,j)_k$:
               * Output $S_n(p,q)_r$;
               * If $S_n(p,q)_r = 1$ then add $(p,q)_r$ to the LSP and output sign of $x_T(p,q)_r$;
               * If $S_n(p,q)_r = 0$ then add $(p,q)_r$ to the LIP;
            – If $\mathcal{L}(i,j)_k \neq \emptyset$ then move $(i,j)_k$ to the end of the LIS as “type-B” entry; else, remove $(i,j)_k$ from the LIS;
      2.2.4. If “type-B” entry and $\alpha_{SD}(T) = 1$:
         • Output $S_n(L(i,j)_k)$;
         • If $S_n(L(i,j)_k) = 1$ then:
            – Add each $(p,q)_r \in \mathcal{O}(i,j)_k$ to the end of the LIS as “type-A” entry;
            – Remove $(i,j)_k$ from the LIS.

3. **Refinement pass**: For each $(i,j)_k \in \text{LSP}$, except those found significant in the current sorting pass, output the $n^{th}$ most significant bit of $|x_T(i,j)_k|$.

4. **Quantization-step update**: Decrement $n$ by 1 and go to Step 2.
Input: set $T$ with root $(i,j)_k$

1. If $(i,j)_k$ is “type-A” entry:
   
   1.1. For each $(p,q)_r \in O(i,j)_k$:
      
      1.1.1. If $\alpha_p(p,q)$ not coded yet then output $\alpha_p(p,q)$;
      
      1.1.2. If $D(p,q)_r \neq \emptyset$ then:
         
         • If $\alpha_{SD}(D(p,q)_r)$ not coded yet then output $\alpha_{SD}(D(p,q)_r)$;
         
         • If $\alpha_{SD}(D(p,q)_r) = 0$ terminate processing of $D(p,q)_r$;
         
         • If $\alpha_{SD}(D(p,q)_r) = 1$ then:
            
            – If $\alpha_{SR}(D(p,q)_r)$ not coded yet then output $\alpha_{SR}(D(p,q)_r)$;
            
            – If $\alpha_{SD}(D(p,q)_r) = 0$ then go to Step 1 treating $D(p,q)_r$ as new “type-A” input;

2. If $(i,j)_k$ is “type-B” entry:

   2.1. For each $(p,q)_r \in O(i,j)_k$, go to Step 1 treating $D(p,q)_r$ as new “type-A” input;

---

Figure 2.16: Shape Code Set (SCS) Subroutine

of the colour channel index $k$ and the use of the CSPIHT SOT structure linking the luminance and chrominance channels [30].

In summary, the ST-SPIHT coder operates the same way as in SPIHT, but includes a number of tests to determine whether trees or individual coefficients are inside or outside the object. The full analysis of the shape coding can be found in [9, 10]

### 2.4 Image Encryption

Image encryption refers to a process in which image data is obscured via a cryptographic algorithm. The goal is to make the image data confidential, with retrieval of the data (decryption) requiring a secret key or some other secret component. Unauthorized individuals who do not have the key should not be able to reveal any portions of the image data. A generic image encryption system is shown in Figure 2.17. The image data input to the image encryption system may be represented in different domains (to be discussed below). Note that this type of system does not address other aspects of security, such as
authentication or integrity.

There is a wide range of applications which rely on or benefit from image encryption technology. These include secure communications, digital rights management (DRM) (such as in pay-TV), and privacy protection. Accordingly, there is a diverse range of image and video encryption methods [4].

As discussed in Section 2.1, most modern applications which incorporate digital imagery must employ image coding in some form. In reality, it is quite rare for an application to not employ image coding somewhere in the pipeline. For this reason, the image encryption techniques are discussed here in the context of their interaction with image coding.

There are a number of technical challenges which must be addressed in designing an image encryption scheme. These are:

**Data Volume and Computational Complexity:** One of the primary concerns that must addressed in many image and video systems is the processing of large volumes of data. It is this large volume that necessitates the use of image coding, as discussed in Section 2.1. For example, video surveillance systems may involve processing multiple streams of data, up to 30 image frames per second, each. At
VGA resolution (640 × 480), each 30 fps stream will be approximately 26 MB/s of data, uncompressed. However, in implementing coding systems, a tradeoff is being made to reduce the data volume at the cost of computational resources. Hence, when encryption is added to a system, the added computational burden must be addressed. Generic data encryption approaches, such as DES and AES [40], demand significantly less resources than coding operations, but it is not negligible. For hardware systems designed to specification, adding encryption functionality requires the addition of CPUs or ASICs. If encryption is performed pre-compression, it will require encrypting a large volume of data, and hence will require the appropriate resources to be implemented. This has lead to so-called “partial” or “selective” encryption approaches which operate during- or post-compression. The idea of effective encryption in the compressed domain was first introduced by Shannon [41]. Conceptually, compressed data is easier to encrypt due to decreased redundancy — i.e., there is less possibility of information leakage. Hence, some selective encryption schemes attempt to encrypt only the most significant portions of coded data which share minimal mutual information with the data that is left unencrypted.

**Effective Confidentiality (Security):** The primary goal of the image encryption system is to completely obscure the data.\(^3\) The nature of visual data, being both subjective and, often, redundant, imposes significant demands on the obscuration process. For example, the low frequency components of an image are often considered the most visually significant (and natural images typically have most energy in the low frequency range). However, if only the low frequency components are obscured, edges would still be present in the image, revealing important image details. This is called information leakage. It is clear that the obscuration processing must take into account a wide variety of possible image content as well as

\(^3\)Some specialized applications, such as privacy protection, require only obscuring certain features. Such approaches are discussed in Section 2.5.
the specific usage scenarios. To address this challenge, the entire image content can be encrypted, but this naive approach does not address the technical challenge associated with data volume and computational complexity.

**Effect on Compression Performance:** As previously stated, we consider encryption systems in the context of an imaging system that will also include image coding. Hence, the effect that the encryption process has on the compression performance must be addressed. If the encryption operation is performed pre-compression and produces a noise-like output, the compression performance will be severely impacted (compression will often become completely ineffective, in this case). Furthermore, lossy coding systems are designed to control visual distortion; with the addition of pre-compression encryption, the nature of the visual distortion present upon decoding, then decrypting, of the data may be unpredictable. Finally, if the encryption system does not take the effect of coding into account at all, applying coding may make decryption impossible, making the data effectively irretrievable. These issues make most pre-compression encryption approaches fundamentally incompatible with image coding systems.

**Implementation Complexity:** Aside from the computational complexity of the encryption operation, the complexity of implementing the encryption system must also be taken into consideration. If the encryption system requires direct integration with a coding system, this may make it less modular and difficult to deploy, especially in legacy systems.

**Bitstream Conformance and System Functionality:** The operation of the larger system in which the encryption is employed must be taken into consideration. If the encryption system alters the bitstream format, it may negatively affect other system components which depend on bitstream conformance. For example, a transcoding server may adapt the coded data for different user requirements. If the encryption
function is implemented post-compression and creates a non-conforming bitstream, it may render the transcoding operation nonfunctional.

In order to address the different technical challenges and application requirements, a number of approaches have been developed that operate in different signal domains. Referring to Figure 2.1, we see that there are three general signal domains\textsuperscript{4} to be considered with respect to image coding systems:

**Pixel Domain:** This is the pre-compression, spatial representation of the image data, where each pixel datum represents the colour of the image at a precise spatial location. Encryption methods that operate in this domain will typically *not* take into consideration the use of any specific image coding system. Pixel domain encryption may be implemented as a module in an imaging system that is not directly related to image coding.

**Transform Domain:** The representation of the image data in the transform domain depends on the particular transform utilized in the coding scheme. The DCT is still the most prevalent transform, providing a frequency domain representation. Upon the reverse transform, manipulations in this domain will be spread across the entire spatial domain block on which the transform was performed (typically $8 \times 8$). With the DWT, a subband representation is provided. The extent of the spread of the manipulations in this domain into the spatial domain depends on the frequency resolution (bandwidth) of the subband being manipulated. Based on the Heisenberg Uncertainty Principle [22], subbands representing narrow frequency bands (high frequency resolution) will result in more spatial domain spread. For the dyadic DWT, it is the lower frequency subbands which offer higher frequency resolution and more spatial domain spread. In general, transform domain manipulations require direct access to the internal operation of the image coding system.

\textsuperscript{4}The figure depicts a wavelet-based coding system, but can be representative of generic transform-based coding schemes, with the replacement of the DWT with another transform, such as the DCT.
**Code Domain:** In the code domain, also known as compressed domain, the image data is represented as code symbols (typically binary, if considering the final output). The code symbols represent abstract constructs that are specific to the coding system employed. For example, in JPEG, the code symbols may be the Huffman codes representing the run length of a set of AC DCT coefficients. For SPIHT, the code symbols may represent the binary significance decisions related to a tree of wavelet coefficients. Aside from the code symbols that directly represent the original image data, additional data such as code tables and header data are included in this domain. Manipulations in the code domain do not necessarily require access to the coding system as they may be performed after coding is complete. However, operating in the code domain typically requires understanding of the code format, and hence may involve parsing of the code stream if the manipulations have not been implemented within the coding system.

Based on the technical challenges and the properties associated with each signal domain, there are a number of advantages and disadvantages to performing encryption in each of three domains. These are outlined in Table 2.2.

In summary, in most cases, pixel domain encryption is considered to be incompatible with coding systems. Transform domain encryption may offer greater compatibility at the cost of increased implementation complexity. Code domain encryption is often the most computationally efficient approach, but may be moderately difficult to implement and may affect bitstream conformance.

The following sections provide overview of the some of the particular image encryption methods, categorized according to their domain of operation.

### 2.4.1 Pixel Domain Encryption

As previously stated, pixel domain encryption approaches generally produce noise-like image data that is fundamentally incompatible with image coding systems. This is best
### Table 2.2: Advantages and disadvantages of encryption in different domains.

<table>
<thead>
<tr>
<th>Domain</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
</table>
| **Pixel** | • Can be independent of coding system, making it simple to implement  
• Can be implemented with spatial precision with the possibility of obscuring arbitrary regions | • Often makes compression impossible by creating noise-like image data  
• Requires encryption of large portion of the data to effectively achieve confidentiality |
| **Transform** | • Can effectively obscure large regions with minor manipulations | • Can be difficult to implement since it requires direct integration with coder  
• May reduce compression performance, depending on how it is integrated with coder |
| **Code** | • Does not affect compression performance  
• Will often require encrypting only a very small portion of the data | • Can be difficult to implement since it requires knowledge of the code syntax  
• May negatively affect bitstream conformance |
illustrated with an example.

Figure 2.18(a) shows the “Lena” test image and Figure 2.18(b) shows a version that has been encrypted by a random permutation. This encryption approach is not completely secure (e.g., the histogram can still be calculated and potentially exploited [42]), but the noise-like nature of the output is common among pixel domain encryption techniques [43]. To test the behaviour of an image coding system with this type of input (i.e., pre-compression, pixel domain encryption), the original image and the encrypted image were both coded using SPIHT [5], using a CDF 9/7 wavelet transform with 5 levels of decomposition. They were decoded at a variety of bit-rates, and the encrypted version was subsequently decrypted by reversing the permutation. The achieved quality (PSNR) vs. bit-rate is shown in Figure 2.19. As is clearly demonstrated, at the test bit-rates (0.1 to 2.0 bpp, corresponding to compression ratios 240:1 to 12:1) the encrypted version results in quality levels that are completely unusable in most applications (in the range 15 to 20 db). Figures 2.18(c) and 2.18(d) shown the decoded original image and the decoded and decrypted image, respectively, at 1.0 bpp. The decrypted image is highly distorted (16.6 dB) while the decoded original image is almost visually indistinguishable from the original (34.6 dB).

Given the inherent incompatibility with coding systems, the existing pixel domain encryption techniques described in the literature are discussed here only briefly. The pixel domain techniques represent some of the earliest approaches to image encryption, having their origins in the analog video scrambling approaches [4, 44]. However, simple permutation approaches, like the example above, are vulnerable to known/chosen plaintext attacks where a permutation mask can be generated, equivalent to the decryption key [42].

One of the popular approaches is to utilize chaotic maps. The approach utilizing 2-D chaotic maps was originally proposed in [45, 46, 47] and refined in [48, 49, 50]. The technique can be summarized as follows [42], for input image of size $M \times N$: 
Figure 2.18: “Lena” test image and encrypted version, before and after SPIHT coding. The pre-compression, pixel domain encryption significantly reduces coding performance.
Figure 2.19: PSNR vs. bit-rate for “Lena” image and encrypted version using random pixel permutation, coded using SPIHT

1. Define a discretized and invertible 2-D chaotic map on an $M \times N$ lattice; the discretized parameters serve as the decryption key

2. Iterate the discretized 2-D chaotic maps on the input image to permute all the pixels

3. Use a cipher to modify all the pixel values to flatten the histogram

4. Repeat the permutation and cipher for $k$ rounds to generate the final encrypted image

Any generic encryption technique utilizing 1-D chaotic maps can be applied to images (applied in the row or column direction). Some of the methods targeted directly for images are proposed in [51, 52, 53].

Another type of approach builds upon the so-called “secret sharing” cryptographic techniques [54, 55]. These \( \{k, n\} \) secret sharing schemes split the data into $n$ shares, each of which does not individually reveal image data. The shares are typically distributed
to separate individuals. This is a modern variation on the original visual cryptography scheme which utilized the human visual system for share recombination [56]. When $k$ or more shares are combined, the image data can be correctly decrypted; with fewer than $k$ shares, the image data remains confidential. The proposed approaches in [54, 55] operate on the most-significant bit-planes of the pixel values, hence can be considered a type of selective encryption technique.

Another selective encryption method which is applied to the most significant bit-planes of the pixel values is proposed in [57]. The AES cipher was used to encrypt the bit-plane values. Alternatively, the approach proposed in [58] involved encrypting the least significant bit-planes. This required the encryption of many bit-planes to sufficiently obscure the visual data.

2.4.2 Transform Domain Encryption

The transform domain encryption techniques typically utilize either the DCT or the DWT. The methods that encrypt in the DCT domain are usually designed to be integrated with the DCT based coders, such as MPEG-1/2/4 [19, 20, 21] and JPEG [15]. The methods that encrypt in the DWT domain are usually designed to be integrated with the DWT based coders, such as JPEG2000 [16], EZW [28], and SPIHT [5]. There is a class of optical algorithms which operate in the Fourier domain [59], but since these are applied to analog images, they are beyond the scope of the discussion here.

The DCT domain encryption methods typically involve encrypting one or more of the lowest frequency components in each of the transform blocks, which are typically $8 \times 8$ in size. The difficulty with this approach is determining how many components need to be encrypted to ensure confidentiality. This is illustrated in Figure 2.20. The original “Lena” image was first transformed from RGB to YCbCr, and then the DCT was applied to $8 \times 8$ blocks in each channel. The first $n$ DCT components (out of 64) were discarded, according to zig-zag ordering [15], and the image reconstructed to visually reveal the
data that remains unencrypted. Referring to Figure 2.20, each row of images represents
the scenario of \( n = 10, 32, \) and 50 coefficients, respectively. The first column shows the
data that would be encrypted (retaining components 1 to \( n \)); the second column shows
the data that remains unencrypted (components \( n + 1 \) to 64); the third column is the
same image as in the second column, after automatic contrast enhancement. It can be
seen clearly that even after discarding the lowest 50 frequency components, there is still
significant data in components 51 to 64 that reveal image features (mainly edges). Hence,
it is problematic to devise a selective encryption scheme in the DCT domain.

In practice, when integrated with a coding system like MPEG or JPEG, many of the
high frequency components will have been quantized to 0. However, if all of the high
frequency components that are to be left unencrypted are quantized to 0, the encryption
system is effectively encrypting all of the available data. If there is some high frequency
data that is not discarded during quantization and remains unencrypted, then there is
the risk that important image details will be revealed (information leakage).

It should be noted that in some applications, mainly DRM in consumer video, the
application does not require complete confidentiality. The goal is simply to make the
content unpleasant to view. In which case, it may be acceptable to reveal the details as
shown in Figure 2.20.

One of the earliest methods of performing transform domain encryption is the SECMPG
system [60]. Targeting MPEG-1, this offered multiple levels of security. At the first level,
it encrypted only header information, making it effectively a code domain approach. At
the second level, the first 4 to 9 DCT coefficients are encrypted. At the third level, it
encrypted the entire DCT blocks. At the forth level, it encrypted the entire bit-stream.
The proposed approach utilized the DES [40] cipher to perform the actual encryption. It
is expected that this method will reduce the coding efficiency since the coefficients values
are changed before entropy coding.\(^5\)

\(^5\)The effect of this scheme on coding efficiency has not been discussed in any known literature.
Figure 2.20: Reconstruction of only partial DCT coefficients. First column contains low frequency components 1 to \( n \) (to be encrypted); second column contains high frequency components \( n + 1 \) to 64 (to not be encrypted); third column contains contrast enhancement of column 2. First row: \( n = 10 \); second row: \( n = 32 \); third row: \( n = 50 \).
Another approach [61] involves a random permutation of the zig-zag DCT coefficient ordering utilized in MPEG-1 and JPEG. However, the permutation approach was found to be weak against known plaintext attacks.

The approach in [62, 63] involves the random sign inversion of AC coefficients. This provides moderate confidentiality (some details are visible), with low impact on coding performance. To achieve greater confidentiality, the DC values could be scrambled as well, but this affects coding performance more significantly. Their scheme was targeted toward privacy applications, hence only the DCT blocks corresponding to certain regions of interest were encrypted.

Some other methods which encrypt the DCT coefficients in different ways are proposed in [64, 65, 66].

In the case of the DWT domain, the issue of data leakage is similar to the DCT. This is illustrated in Figure 2.21. The image shown was reconstructed based on the three first level, high frequency subbands (utilizing the CDF 9/7 DWT), HL1, LH1, and HH1. This represents the unencrypted data if the first level, low frequency subband (LL1) is encrypted. Since this represents the largest subband of low frequency data (further decomposition will result in a more narrow-band LL subband), it represents the “best case” scenario for partial encryption in the wavelet domain, targeting low frequency subbands. Hence, partial encryption directly in this domain would generally result in significant information leakage.

In [67], as a means of fast encryption, random permutations of wavelet coefficients before coding using SPIHT or JPEG2000, was studied. However, it was found that this significantly reduced compression performance (up to 27% for test images).

The scheme in [68] is a code domain encryption scheme, but the content targeted relates directly to the two lowest-frequency levels of the DWT decomposition in the SPIHT coding scheme. This approach is problematic because: a) the number of DWT decomposition levels is generally a user controllable parameter, so the frequency bands
Figure 2.21: Reconstruction of “Lena” image based on the first level, high frequency DWT subbands (HL1, LH1, and HH1), with automatic contrast enhancement.

corresponding to the two lowest decomposition levels is variable; and b) as shown in 2.21, the higher frequency subbands will leak information.

2.4.3 Code Domain Encryption

The final domain in which encryption may be considered is the code domain. Code domain encryption schemes may be integrated with the coder or implemented as a code stream post-processor. In the latter case, the encryption scheme will typically be required to parse the code stream at some level. This approach has several key advantages over the other domains:

- The coding performance will be unaffected because the manipulations are being performed post-compression
- The encryption is being performed in domain in which the data has been largely reduced in volume
- There is more opportunity for selective encryption approaches (compared to the pixel and transform domain in which information leakage is difficult to avoid)
Considering the last point, it should be noted that code domain selective encryption is still not trivial. There are still many complex, interacting components which may reveal information about the parts that are encrypted.

The composition of the code domain data is highly dependent on the particular coding standard being considered. For most of the standards of interest (e.g., JPEG and MPEG-1/2/4), the code stream consists largely of header information, code tables, and entropy code symbols relating to the actual image data. The most common entropy coder employed is the Huffman coder [11]. For the JPEG standard, the entropy coded data is derived from run-length coded DCT AC coefficients, and the differentially coded DC coefficients. For the MPEG standards, compared to JPEG, there is significantly more header data as it relates to the hierarchy of blocks, macroblocks, slices, frames, and groups of pictures. There is also the inclusion of motion vectors for inter-frame prediction. The coded DCT data is represented in a similar manner to JPEG.

A number of schemes have been proposed that encrypt different portions of the JPEG or MPEG code data [3]. Some schemes may only encrypt header metadata [60, 69]. Others target different combinations of the coded data [70, 71, 72]. However, due to the relationship between code components within frames and between frames in video, weaknesses have been found with almost all the schemes [42].

Of particular interest is the scheme proposed in [68]. Using SPIHT with monochrome images, it was proposed that only the significance bits (both for individual coefficients and trees) of the top two tree levels, along with the initial quantization threshold, be encrypted. The number of bits encrypted is variable depending on the image content. Monochrome test images of size $512 \times 512$ for screen-display applications were encoded at 0.8 bpp and it was found that less than 2% of the coded source was typically encrypted. It was claimed that confidentiality is achieved not just through securing the most significant information, but by making the correct state of the decoder difficult to determine.
This approach is flawed for the reasons stated in the previous section (i.e., it targets low frequency subbands which may result in information leakage through the other subbands). However, it does first introduce the concept of affecting the decoder execution path to obscure the data. This concept will be expanded upon in the secure coder proposed in this thesis.

### 2.5 Privacy Protection in Video Surveillance

Privacy in video surveillance has become a topic of significant interest as CCTV installations have become pervasive in public and private environments [6]. Most currently deployed systems indiscriminately capture visual data of various subjects and objects which may be considered private. For example, the identity of most subjects in video surveillance is not required when they are not involved in unauthorized activities. However, while they are in the camera field of view, their identity as well as their activities and behaviour will be captured and stored. In many cases, there will not be any significant protection of this data, leaving it open for abuse. Stored data can be cross-matched to create activity profiles of unsuspecting individuals. Or the data may be used for simple voyeurism.

A number of researchers have attempted to address the issue of privacy in video surveillance, but there is no common understanding of what a privacy protection system should accomplish. For example, some approaches attempt to irreversibly remove (obscure) private data only when it is viewed [8] (e.g., by security guards monitoring live). This generally requires that the original data be retained in case the obscured data needs to be examined (e.g., for legal purposes). This may be considered a weak form of privacy protection since it does not address the issue of abuse of stored data.

On the other hand, other approaches attempt to encrypt the private data to store it in protected form and only allow retrieval (via release of a key) when required for
Regardless of general approach, a common constraint imposed on all systems is that they do not impede the stated purpose of the surveillance system, which is usually to maintain security of a physical space. The technical challenges associated with designing and deploying an effective privacy protection system for video surveillance can be summarized as follows:

- Defining what is “private information” in such a manner that the private information can be distinguished from the non-private information in video signals
- How private information can be separated or segmented from the non-private information in video signals
- How the private data can be protected

A generalized system for privacy protection in video surveillance systems is shown in Fig. 2.22. In this context, privacy protection is defined as the securing of private information from intentional or accidental access by persons who do not have a legitimate, authorized use for the data.

Some of the work in this area has focussed on the higher level system design and the object detection/tracking. The computer vision approach of [73] provides three policy-dependent options to hiding privacy data: summarization; transformation (obfuscation);
and encryption. In the case of encrypted output, traditional encryption is applied to the entire private data stream, which may be computationally infeasible in many multi-stream video surveillance systems.

The work in this thesis addresses the issue of protection/obscuration; the design and implementation of the other parts of the system, such as the object detection/tracking are beyond the scope of the discussion here. As with image (frame) encryption technique discussed in Section 2.4, private visual data protection techniques which involve encryption can be evaluated based on the domain in which they operate.

The first class of private data obscuration approaches involves scrambling, obscuring, or masking techniques to protect the identity of human subjects in video surveillance [74, 8, 75, 76]. In these schemes, the visual texture data of the subject’s face or whole body are discarded or irreversibly transformed and hence can be performed pre-compression and designed in such a manner that it does not negatively affect compression (i.e., the obscured data does not have to appear noise-like). In [74], the subject’s body image is masked, revealing only a silhouette. However, such a silhouette may still allow identification of the subject via biometric modalities such as gait [77]. Similarly, in [8], the focus is on removing appearance information while retaining structural information about the body in order to assess behavior. The approach in [75] is to “de-identify” face images so that facial recognition software cannot be used to reliably identify the subject, but enough facial features remain so that the image could still be used for detecting behavior. In this so-called $k$-Same approach, face images are clustered based on a distance metric, and the images replaced by a representative image generated by averaging of components based on pixels or eigenvectors. This approach, however, does not obscure the whole body image, and again, the original data is discarded and cannot be retrieved by authorized users. In [76], coloured markers are worn by subjects who wish to have their face obscured in a particular surveillance environment. Employing AdaBoost to learn the marker’s colour model and Particle Filtering to track the marker from frame-
to-frame, the subject is tracked in real-time and an elliptical mask placed over the head region. However, the scheme is not practical in public scenarios as it requires subjects to “opt-out” through the use of the coloured marker.

Another class of privacy protection schemes attempts to separate private features from the input signal and secure them in a fashion so that they may still be retrieved for future use (e.g., via encryption/decryption) \[78, 73, 79, 62\]. In [78], a region of interest (ROI) is defined for face data within a frame, and the corresponding coefficients downshifted in order to be coded and protected in a separate quality layer using Motion JPEG 2000 \[80\]. However, using a traditional, non-shape-adaptive wavelet transform, the wavelet domain separation of ROI content only allows for rough separation of content in the spatial domain, thus disallowing precise object vs. background separation possible with object-based coding. The scheme proposed in [79] embeds the private information of subjects as an encrypted watermark within the surveillance frames. However, the private data is limited to rectangular regions of the image frame and the utilization of traditional encryption and watermarking may be computationally burdensome. In [62], a reversible wavelet-domain scrambling is performed on ROI-defined private data, thus allowing subsequent retrieval of the private data by authorized users. This approach, as in [78], does not allow explicit spatial domain separation of the object of interest and the background, and the region-of-interest shape is not secured. Furthermore, the scrambling is performed before compression, in the transform domain, resulting in a modest reduction in coding performance \[62\].

The previously mentioned scheme in [63] involves scrambling of AC DCT coefficients. First, DCT blocks which spatially correspond to the private data are identified, and the scrambling is applied only to those blocks. The scrambling operation involves random sign inversion and reduces coding performance. This does not obscure all the relevant visual content, but provides a generally “scrambled” look. To achieve greater security, the DC coefficients could be scrambled as well, but this significantly reduces compression
2.6 Chapter Summary

This chapter covered the background material related to wavelet-based image coding and object-based. The fundamental concepts are necessary for the understanding of the secure coding techniques presented in following chapters.

Additionally, the prior art on image encryption and privacy protection in video surveillance was discussed. The encryption techniques were classified based on their domain of operation: pixel, transform, or code. An extensive discussion on the implications of operating in each domain was provided. The schemes were evaluated based on their interaction with coding systems (a necessary component of almost all imaging systems). It was deemed that code domain encryption is the most promising because it will generally not adversely affect coding performance.
Chapter 3

Secure Image Coder

The proposed wavelet-based secure image coder, called SecSPIHT, is a colour image encryption scheme built upon the SPIHT coding algorithm [5]. SecSPIHT has the following key features:

- Complete confidentiality; image is completely obscured and can only be decoded and decrypted with the provision of a secret binary key

- Encryption is performed in the code domain — coding performance of SPIHT is maintained

- Very computationally efficient; employs intelligent selective encryption which encrypts a very small portion of the code stream

- Bitstream maintains full scalability — it can be arbitrarily truncated to provide rate-distortion scalability

- Fully reversible; any loss is from coding (compression) operation, not from encryption

- Can be operated in a lossless fashion if integer colour transform and DWT are utilized
Chapter 3.  Secure Image Coder

This chapter is organized as follows. Section 3.1 provides some important definitions and describes the methodology employed in the design of the secure coder. Section 3.2 provides the complete algorithm description. Section 3.3 provides a security analysis of the proposed scheme. Section 3.4 provides some simulation results on sample images. Finally, the chapter concludes with a summary in Section 3.5.

3.1 Definitions and Methodology

The SPIHT coder is unique in that the execution path of the encoder and decoder are completely data dependent. The proposed secure coding approach involves the encryption of key bits in the output bit-stream which the decoder needs to correctly interpret to follow the correct execution path.

We denote SPIHT bit-stream as the ordered set of bits $B$. The bit-stream can be divided into the ordered subsets $B = \{B_{n_{\text{max}}}, B_{n_{\text{max}}-1}, B_{n_{\text{max}}-2}, \ldots\}$, where $B_n$ is the set of bits obtained during coding iteration for bit-plane $n$ (i.e., representing the value $2^n$), and $n_{\text{max}}$ is the highest bit-plane at which coding is initiated. Each $B_n$ can be further subdivided into $B_n = \{B_{n,\text{LIP}}, B_{n,\text{LIS}}, B_{n,\text{LSP}}\}$, where $B_{n,\text{LIP}}$ denotes the ordered set of bits obtained during the first phase of the sorting pass where coefficients in the LIP are tested for significance; $B_{n,\text{LIS}}$ denotes the ordered set of bits obtained during the second phase of the sorting pass where entire trees are tested for significance; and $B_{n,\text{LSP}}$ denotes the ordered set of bits obtained during the refinement pass. This decomposition of the bit-stream is shown pictorially in Figure 3.1.

Each set of bits $B_{n,\text{LIP}}$ is composed of significance bits ($B_{n,\text{LIP}-\text{sig}}$) and sign bits ($B_{n,\text{LIP}-\text{sgn}}$). Similarly, each set of bits $B_{n,\text{LIS}}$ is composed of significance bits ($B_{n,\text{LIS}-\text{sig}}$) and sign bits ($B_{n,\text{LIS}-\text{sgn}}$) for individual coefficients, and significance bits for trees ($B_{n,\text{LIS}-\text{Tsig}}$). This decomposition of the bit-stream with example bits is shown in Figure 3.2. The interpretation of these bit classes is shown in Table 3.1.
Figure 3.1: SPIHT bit-stream

Figure 3.2: Composition of subset $B_n$ of SPIHT bit-stream
Chapter 3. Secure Image Coder

<table>
<thead>
<tr>
<th>Bit Type</th>
<th>Coding Phase</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( B_{n,LIP-sig} )</td>
<td>LIP</td>
<td>Significance bits for individual coefficients</td>
</tr>
<tr>
<td>( B_{n,LIP-sgn} )</td>
<td>LIP</td>
<td>Sign bits for individual coefficients</td>
</tr>
<tr>
<td>( B_{n,LIS-sig} )</td>
<td>LIS</td>
<td>Significance bits for individual coefficients released during partitioning</td>
</tr>
<tr>
<td>( B_{n,LIS-sgn} )</td>
<td>LIS</td>
<td>Sign bits for individual coefficients released during partitioning</td>
</tr>
<tr>
<td>( B_{n,LIS-Tsig} )</td>
<td>LIS</td>
<td>Significance bits for whole trees</td>
</tr>
<tr>
<td>( B_{n,LSP} )</td>
<td>LSP</td>
<td>Refinement bits</td>
</tr>
</tbody>
</table>

Table 3.1: SPIHT bit classification

Based on the understanding of the SPIHT algorithm (see Section 2.2.3), it is clear that the classification of each bit is dependent on the classification of the previous bit and the value of the current bit. We use \( b^j_{n,LIP} \) to denote the \( j \)th bit in the set \( B_{n,LIP} \), for \( j = 0, 1, 2, \ldots, N_{n,LIP} - 1 \), where \( N_{n,LIP} \) is the total number of bits in \( B_{n,LIP} \). It is known \textit{a priori} that the first bit is a significance bit:

\[
 b^0_{n,LIP} \in B_{n,LIP-sig}. \tag{3.1}
\]

However, the classification of the second bit depends on the value of \( b^0_{n,LIP} \):

\[
 b^1_{n,LIP} = \begin{cases} 
 B_{n,LIP-sig} & \text{if } b^0_{n,LIP} = 0 \\
 B_{n,LIP-sgn} & \text{otherwise} 
\end{cases} \tag{3.2}
\]

This can be generalized for \( 0 \leq j < N_{n,LIP} \) as follows:

\[
 b^{j+1}_{n,LIP} = \begin{cases} 
 B_{n,LIP-sgn} & \text{if } b^j_{n,LIP} \in B_{n,LIP-sig} \text{ and } b^j_{n,LIP} = 1 \\
 B_{n,LIP-sig} & \text{otherwise} 
\end{cases} \tag{3.3}
\]

This is clearly a first order (memoryless) Markov process with two states, with the bits \( b^j_{n,LIP} \) representing coded state transition instructions. At bit \( j \) during the LIP coding phase, we define the state \( S^j \) of the coder/decoder as follows:

\[
 S^j = \begin{cases} 
 S_{LIP-sig} & \text{if } b^j_{n,LIP} \in B_{n,LIP-sig} \\
 S_{LIP-sgn} & \text{if } b^j_{n,LIP} \in B_{n,LIP-sgn} 
\end{cases} \tag{3.4}
\]
Based on Equations 3.3 and 3.4, the next state, $S_{j+1}$, is dependent on the previous state and the value of the previous bit, as follows:

$$S_{j+1} = \begin{cases} 
S_{\text{LIP-sgn}} & \text{if } S_j = S_{\text{LIP-sig}} \text{ and } b_{n,\text{LIP}}^j = 1 \\
S_{\text{LIP-sig}} & \text{if } (S_j = S_{\text{LIP-sig}} \text{ and } b_{n,\text{LIP}}^j = 0) \text{ or } S_j = S_{\text{LIP-sgn}} 
\end{cases}$$

We can see that the value of $b_{n,\text{LIP}}^j$ only affects the state of the decoder when $S_j = S_{\text{LIP-sig}}$ (i.e., $b_{n,\text{LIP}}^j \in B_{n,\text{LIP-sig}}$). When $S_j = S_{\text{LIP-sgn}}$, the next state is always $S_{\text{LIP-sig}}$, regardless of the value of $b_{n,\text{LIP}}^j$. This is illustrated in the graph in Figure 3.3.

The importance of the state $S_j$ of the decoder is twofold. First, as already shown, the state determines how the value of the current bit should be interpreted in terms of determining the next state of the decoder. Second, the state determines how the value of the current bit should affect image reconstruction at the decoder. If $S_j = S_{\text{LIP-sig}}$, the current bit determines if the next coefficient in the queue has its MSB in the current bit-plane $n$; if $S_j = S_{\text{LIP-sgn}}$, the current bit determines the sign of the coefficient currently being processed.

Since only the bits $b_{n,\text{LIP}}^j \in B_{n,\text{LIP-sig}}$ control the state of the decoder, keeping these values secret will prevent the decoder from properly executing. The proposed secure coder exploits this fact.

During the LIS coding phase (coding and partitioning of trees), the situation is more
Figure 3.4: Graph of Markov process during SPIHT LIS coding/decoding phase
complicated. Using the same notation as before, there are three states: $S_{\text{LIP}-\text{sig}}$, $S_{\text{LIS}-\text{sig}}$, and $S_{\text{LIS-Tsig}}$ (refer to Table 3.1 for interpretation of corresponding bit classes). The Markov process has memory in this case since individual coefficients are released four at a time during tree partitioning. This is illustrated in Figure 3.4. There is also the classification of the trees as either Type “A” or Type “B” (see Section 2.2.3), which must be taken into account.

As with the LIP phase, the sign bits $b^j_{n,\text{LIS}} \in B_{n,\text{LIS-\text{sgn}}}$ do not affect the decoder execution path. The individual coefficient significant bits $b^j_{n,\text{LIS}} \in B_{n,\text{LIS-\text{sig}}}$ always affect the decoder execution path. The tree-level significant bits $b^j_{n,\text{LIS}} \in B_{n,\text{LIS-Tsig}}$ affect the execution path when the tree is Type “A”; they do not affect the execution path when the tree is Type “B”. This is due to the fact that partitioning of Type “A” trees releases individual coefficients; this is not the case for Type “B” trees. Again, the bits that affect the decoder execution path will be exploited by the propose secure coder.

Finally, examining the LSP coding phase, it can be seen that there is no Markov process or data dependent execution path. This phase involves direct bit-plane coding of all the coefficients in the LSP (found significant at previous bit-planes).

### 3.2 SecSPIHT Algorithm

The proposed SecSPIHT secure image coding system is shown in Figure 3.5. The system follows the generic wavelet-based coding approach (see Section 2.2), but utilizes a secure coding component (the actual SecSPIHT coder) which accepts a secret binary encryption key, $k_E$, to produce a confidential bitstream. There is a matching secure decoding component which accepts a secret binary decryption key, $k_D$, which is required to correctly decode the data.

Using the same representation as in Section 2.1, the input is an $M \times N$ colour image
Figure 3.5: SecSPIHT secure image coding and decoding system

of the form

\[ \mathbf{x} : \mathbb{Z}^2 \rightarrow \mathbb{Z}^3, \quad (3.6) \]

representing a two-dimensional matrix of three-component RGB (red, green, blue) colour samples

\[ \mathbf{x}(i,j) = [x(i,j)_1, x(i,j)_2, x(i,j)_3], \quad (3.7) \]

with \( i = 0, 1, ..., M - 1 \) and \( j = 0, 1, ..., N - 1 \) denoting the spatial position of the pixel, and \( x(i,j)_k \) denoting the component in the red \((k = 1)\), green \((k = 2)\), or blue \((k = 3)\) colour channel.

For lossy coding, the ICT colour transform is utilized (see Section 2.2.1). In this case, the output pixel values will be real. For lossless coding, the RCT colour transform is used, producing integer output.

The DWT utilizes the CDF 9/7 wavelet filters (see Appendix B) for lossy coding. For lossless coding, the Le Gall 5/3 integer filters are used. The output is the transformed image, \( \mathbf{x}_T \).

The SecSPIHT coder is shown in more detail in Figure 3.6. It is a selective encryption scheme, where individual bits of the SPIHT bit-stream are encrypted. The key components are the SPIHT coder and an encryption function, \( f_E(b, k_E) \), which encrypts
individual bits \( b \), based on the encryption key \( k_E \). Fundamental to the SecSPIHT coder design is the selection of which bits to encrypt; the set of encrypted bits, \( B_e \), is defined as:

\[
B_e = \{ B_{n,LIP-sig}, B_{n,LIS-sig} \}, \quad \forall n > n_{\text{max}} - K
\]  

(3.8)

where \( K \) is an input parameter, controlling how many coding iterations during which encryption is performed. These bits are selected because they directly control the execution path of the decoder (see Section 3.1). By making them secret, the decoder cannot correctly decode any of the coded data. Security analysis is provided in Section 3.3.

The SecSPIHT decoder is shown in Figure 3.7. It utilizes a decryption function, \( f_E(b, k_D) \) which reverses the encryption operation of \( f_E(b, k_E) \), using the secret decryption key \( k_D \). For the decoder to be able to execute correctly, it must be able to decrypt the bit-stream one bit at a time in order to determine whether the next bit requires decryption or not (i.e., to determine whether \( b^{i+1} \in B_e \), based on the current decoder state and the value of \( b^i \)). For example, it is known a priori that the first bit is encrypted is always encrypted — i.e., \( b^0_{n_{\text{max}}.LIP} \in B_{n_{\text{max}}.LIP-sig} \in B_e \), and \( S^0 = S_{\text{LIP-sig}} \). This bit must be decrypted and its value evaluated to determine if the next bit is encrypted or not. Referring to Figure 3.3 and Equation 3.5, if \( b^0_{n_{\text{max}}.LIP} = 0 \), then \( S^1 = S_{\text{LIP-sig}} \) and the next
The SecSPIHT scheme does not specify a particular encryption cipher to be used for \( f_E(\cdot, \cdot) \) and \( f_D(\cdot, \cdot) \). However, it does place certain restrictions on the choice. As previously stated, the decryption must be performed bit-by-bit. This necessitates the use of a bit-level stream cipher. If a block cipher were used, it would not be possible to decrypt the stream in an incremental fashion. The common RC4 stream cipher may be used [40]. However, many weaknesses have been found in RC4 — e.g., [81]. Alternatively, the eSTREAM project has many new stream ciphers being developed [82].

The nature of the keys, \( k_E \) and \( k_D \) will be determined by the choice of the stream cipher. In most cases, a symmetric key system will be used, in which case \( k_D = k_E \).

The complete coding algorithm is shown in Figure 3.8. For simplified notation, we introduce the controlled encryption function \( f_{cE}(b, k_E, n, K) \), defined as follows:

\[
f_{cE}(b, k_E, n, K) = \begin{cases} 
    f_E(b, k_E), & n > n_{\text{max}} - K \\
    b, & \text{otherwise}.
\end{cases}
\] (3.9)

Hence, the encryption function is only activated for the first \( K \) iterations of the coding algorithm, after which the input bits are passed through, unencrypted. The choice of the parameter \( K \) is discussed in the next section.
Input: $x_T$, $K$, $k_E$

1. **Initialization:** Find initial quantization level $n = \lfloor \log_2(\max_{i,j}\{|x_T(i,j)|\}) \rfloor$; set LSP as an empty list, and add all $(i,j) \in \mathcal{H}$ to the LIP and those with descendants to the LIS as type A entries.

2. **Sorting pass:**
   
   2.1. For each entry $(i,j)$ in the LIP:
      
      2.1.1. Output $f_{cE}(S_n(i,j), k_E, n, K)$;
      
      2.1.2. If $S_n(i,j) = 1$ then move $(i,j)$ to the LSP and output the sign of $x_T(i,j)$;

   2.2. For each entry $(i,j)$ in the LIS:
      
      2.2.1. If type A entry:
         
         • Output $S_n(\mathcal{D}(i,j))$;
         • If $S_n(\mathcal{D}(i,j)) = 1$ then:
            
            – For each $(k,l) \in \mathcal{O}(i,j)$:
               
               * Output $f_{cE}(S_n(k,l), k_E, n, K)$;
               * If $S_n(k,l) = 1$ then add $(k,l)$ to the LSP and output sign of $x_T(k,l)$;
               * If $S_n(k,l) = 0$ then add $(k,l)$ to the LIP;
            
            – If $\mathcal{L}(i,j) \neq \emptyset$ then move $(i,j)$ to the end of the LIS as type B entry; else, remove $(i,j)$ from the LIS;

      2.2.2. If type B entry:
         
         • Output $S_n(\mathcal{L}(i,j))$;
         • If $S_n(\mathcal{L}(i,j)) = 1$ then:
            
            – Add each $(k,l) \in \mathcal{O}(i,j)$ to the end of the LIS as type A entry;
            
            – Remove $(i,j)$ from the LIS.

3. **Refinement pass:** For each entry in the LSP, except those found significant in the current sorting pass, output the $n^{th}$ most significant bit of $|x_T(i,j)|$;

4. **Quantization-step update:** Decrement $n$ by 1 and go to Step 2.

---

Figure 3.8: SecSPIHT coder algorithm
As with the original SPIHT codec, the SecSPIHT coding operation can be terminated at any point based on a particular quality or bit-rate criterion. The encryption function does not in any way affect the compression performance or the embedded nature of the bit-stream. If an integer processing pipeline was implemented, the SecSPIHT codec operation can be terminated at the end of coding bit-plane $n = 0$ to achieve lossless coding.

The SecSPIHT decoder is identical to the coder, except that bits are read from the stream rather than written, decrypted when necessary, and instead of performing significance tests, the DWT reconstruction is updated accordingly based on the values read from the bit-stream.

### 3.3 Security Analysis

Figure 3.9 shows example output from SecSPIHT, using the “Lena” test image (see Figure 2.2 for original input). The image was coded with a 5-level DWT; it is decoded at 1.0 bpp, with $K = 2$ (i.e., two coding iterations secured). This results in 990 bits encrypted, which is only 0.38% of the total code stream. The HC-128 stream cipher is used for the encryption function [83]. Figure 3.9(a) shows the decoded/decrypted output when the correct decryption key is used. The image presents no degradation due to the encryption functionality; the PSNR is 34.6 dB (the same as without the encryption functionality). Figure 3.9(b) shows the decoded/decrypted output when the incorrect decryption key is used. Clearly, no image details are revealed; the PSNR is 8.1 dB.

From a visual perspective, the SecSPIHT scheme is secure. However, visual evaluation does not guarantee the confidentiality of the data. It may be possible to reveal some or all of the data via simple, intelligent attacks (e.g., in Figure 2.20, a contrast enhancement was required to reveal the image details in some cases). For SecSPIHT, we must evaluate its operation and devise logical attacks to determine its robustness.
For this security analysis, we assume that the chosen stream cipher is perfectly secure. That is, the encrypted bits cannot be revealed by exploiting some property of the cipher itself; it would require a brute force attack (exhaustive search) on the original key itself. This may not be the case in reality, but it is beyond the scope of the discussion here to study cryptanalytic attacks on various ciphers. In this context, we use the term $B$-bit security to refer to the security achieved by a perfectly secure cipher with a $B$-bit random key. In most cases, the ciphertext (i.e., encrypted content) is larger than the key, hence it makes sense to “guess” the key rather than the content. Thus, a successful attack would require an exhaustive search on a key space of size $2^B$. However, in general, for content of size $B_c$-bits, encrypted with a perfectly secure cipher with a $B_k$-bit key, security of $\min(B_k, B_c)$ bits security is achieved.

Here we consider a number of possible attacks to access the image data.
3.3.1 Attack I – Brute Force Attack on $B_e$

The analysis provided in Section 3.1 makes it clear that for the decoder to execute correctly, it must have access to the original values of the encrypted bits $B_e$. These bits may be attacked directly via brute force (exhaustive search). However, this attack approach is complicated by the fact that the encrypted bits are interspersed with unencrypted bits $B_{n,LIP-\text{sgn}}$, $B_{n,LIS-\text{sgn}}$, $B_{n,LIS-T\text{sig}}$, and $B_{n,LSP}$ from the same coding iteration. Hence, the attacker must consider not just all the possible bit combinations for $B_e$, but various sizes and positions of $B_e$ within the overall code stream. This may increase the space that must be searched. However, this space is highly dependent on the choice of $K$. Looking at the results in Tables 3.2 to 3.5 (Section 3.4, on pages 80 to 82), we see that it is common for the first coding iteration to find no (or very few) trees that have significant coefficients. The interpretation of this is that only the LL$_Y$ (luminance) subband contains significant coefficients at the highest bit-plane (as might be expected due to the energy compaction properties of the DWT). This means that the encoder never reaches the state $S_{n,LIS-sig}$. Hence, it is only during the initial LIP phase during which any encryption occurs (for bits $B_{n,LIP-sig}$). In this case, it will be known that $|B_e| = |B_{n,LIP-sig}| = |\text{LL}_Y|$, where $|\text{LL}_Y|$ is the number of coefficients in the LL$_Y$ subband, $|B_e|$ is the number of bits in $B_e$, and so on. For a 512 × 512 image, and a 5-level DWT, $|\text{LL}_Y| = 256$. These encrypted bits will be interspersed in $B_{n,LIP}$, amongst $B_{n,LIP-\text{sgn}}$. The size $|B_{n,LIP-\text{sgn}}|$ (and $|B_{n,LIP}| = |B_{n,LIP-sig}| + |B_{n,LIP-\text{sgn}}|$) will depend on how many coefficients are found significant (i.e., each significant coefficient produces a sign bit). Looking again at Tables 3.2 to 3.5, we see that the number of sign bits ranges from 30 to 147 (out of 256, or 257 in one case, in $B_{n,LIP}$) for the test images. This offers a relatively large range of possibilities, however, it will be tied to the nature of the image. The “Document” image, which produced the largest number of sign bits (147) in the first iteration includes many large, blocky features which will produce high magnitude coefficients in the LL subband. If the attacker knows the type of image (e.g., a document), it may be possible to reduce
the search space within $B_{n,LIP}$, for this case of $K = 1$.

For example, the attacker may assume that $|B_e| = 256$. If the attacker can make a good guess of $\beta = |B_{n,LIP-sgn}|$ based on the content type, then the number of possible configurations of $B_{n,LIP-sig}$ (encrypted bits) in $B_{n,LIP}$ is

$$\left( \frac{256 + \beta}{256} \right) = \frac{(256 + \beta)!}{256!\beta!}$$

(3.10)

However, for each configuration, the attacker must still perform an exhaustive search on the possible bit values. But, if the configuration is known, the search space will be further reduced by limitations of the bit values in relation to the configuration (e.g., if two significant bits are adjacent, then the first one must be a “0” since it was not followed by a sign bit).\(^1\)

It is clear that with $K = 1$, the state of the encoder/decoder does not vary significantly (the actual data will vary significantly, but the relative configuration of different types of bits will not). While it’s not clear exactly how an effective attack should be formulated (it would require significant information about the image\(^2\)), it is conceivable that there is sufficient information leakage to warrant a prudent approach, requiring $K > 1$. Referring to Tables 3.2 to 3.5, we see considerable variability in the types of bits created for $K = 2$. The total number of encrypted bits, $|B_e|$, ranges between 428 and 990 (out of a total number of bits in the range 1282 to 1913). Considering the proven efficiency of the SecSPIHT coding approach [5], it is reasonable to assume that after this many bits have been generated, the entropy $H(B_e) \sim |B_e|$. In this case, the search space for an attack directly on $B_e$ will be close to $2^{|B_e|}$. Since the cryptographic key length $B_k$ will generally be much less than $|B_e|$ (e.g., 128 bits), it can be said that $B_k$-bit security is achieved for

\(^1\)It should be noted that any bit combination is a valid SecSPIHT syntax. In the case being considered here, we have already fixed the classification — this limits the possible bit values.

\(^2\)Note, this should not be confused with a known plaintext attack. In a known plaintext attack, the goal is to reveal the key (or equivalent information) to be used for decrypting “future” content. As long as the chosen cipher in SecSPIHT is robust against known plaintext attacks, we are not concerned about such attacks since the key cannot be revealed. Our only concern is with the confidentiality of the given image.
the entire image, with respect to a brute force attack on the encrypted bits, $B_e$.

### 3.3.2 Attack II – Progressive Attack on $B_e$

The progressive nature of the SecSPIHT coder leads to concern that the encrypted bits $B_e$ can be attacked progressively. That is, the attacker may “guess” a small initial portion of bits (via exhaustive search), be given visual confirmation of their correctness, and continue to guess the next portion. In the worst case scenario, the encrypted bit-stream could be attacked one bit at a time, with confirmation after each bit. In this case, instead of a search space of $2^{|B_e|}$, the search space is reduced to $2^{|B_e|}$. This is equivalent to $(\log_2(|B_e|) + 1)$-bits of achieved security. For example, the “Lena” test image, with $K = 2$, produced $|B_e| = 990$. With a bit-wise progressive attack, this would be less than 11 bits of security — completely inadequate in most scenarios.

In practice, a single bit of SecSPIHT code data would not provide sufficient visual feedback to confirm its correctness. At a minimum, the data would have to be segmented into portions large enough for confirmation of “desired” imagery. We can consider a worst case scenario in which the attacker is able to decrypt the portion corresponding to the first coding iteration (see discussion above for why this is unlikely). This is equivalent to reconstructing the highest bit-plane of the DWT data. This reconstruction, for the test images, is shown in Figure 3.10 (see Figures 3.11 to 3.12 for original images). Some of the basic image structures are visible in the reconstruction at this level. Assuming $K > 1$ (i.e., at least one additional iteration of encrypted data remains), to continue with the progressive attack, the attacker will have to guess segments of the next subsequent portions of data and perform the reconstruction (i.e., update the image) to confirm whether it is correct or not.

In reality, this is highly infeasible. The confirmation would require very detailed knowledge of the original image, negating the purpose of the encryption in the first place. Furthermore, a single bit error at a certain location renders the remaining data
Chapter 3. Secure Image Coder

Figure 3.10: Reconstruction from first coding iteration (highest bit-plane)
unusable for visual confirmation since it will be effectively scrambled. In light of this, a progressive attack on $B_e$ is not considered to be possible.

### 3.3.3 Attack III – Attack on Unencrypted Code Portion

Considering that the SecSPIHT coder leaves a large portion of the data unencrypted (for coding iterations corresponding to bit-planes $n \leq n_{\text{max}} - K$), it is logical to attempt to access this potential leak of crucial image data. Since any bit sequence is a legitimate SecSPIHT code stream, it is not generally possible to analyze the data to determine where coding iterations begin and end. There are no apparent long or short term patterns. As previously mentioned, the efficiency of the SecSPIHT coding approach implies that each code bit has entropy near unity. Hence, the code stream would appear random.

As such, the location of the beginning of the unencrypted code portion, starting at $n = n_{\text{max}} - K$, would have to be searched. For $K = 2$, this ranged between 1282 to 1913 for the test images (see Tables 3.2 to 3.5, on pages 80 to 82). However, even if this unencrypted data could be exactly located, the correct state of the decoder at that point would be completely unknown. This includes which coefficients have been found significant, which trees have been partitioned, etc. The result is that the unencrypted data could not be interpreted in any meaningful way. This would require knowledge of the encrypted portion, reducing this attack to the same as Attack I above. Accordingly, if $K > 1$, this attack is considered infeasible.

### 3.3.4 Summary of Security Analysis

Three attacks were considered: i) a brute force attack on the encrypted bits $B_e$; ii) a progressive attack on $B_e$; and iii) an attack on the unencrypted portion of the data. Based on simulation data (from Section 3.4), it was determined that the parameter $K$, controlling the number of encrypted coding iterations, should generally be set $K > 1$ to achieve sufficient variability of the correct decoder state across the encrypted portion.
Under this condition, all attacks were considered infeasible. If a $B_k$-bit random binary key is used with a perfectly secure stream cipher in the SecSPIHT coder, then $B_k$-bit security is achieved for the entire coded image data.

### 3.4 Simulation Results

This section provides some simulation results, applying the SecSPIHT secure coder to a selection of test images. The SecSPIHT coder and decoder system were implemented in MATLAB and C code. All results, except where stated, utilized the lossy pipeline with the ICT colour transform (see Section 2.2.1), and the CDF 9/7 DWT with 5 levels (see Section 2.2.2 and Appendix A). Where the lossless pipeline was utilized, the RCT colour transform and the Le Gall 5/3 integer DWT were used.

Figures 3.11 to 3.12 show the test images, along with the encrypted version, decoded and decrypted with the incorrect key, and with the parameter $K = 2$ (i.e., encryption performed over two coding iterations). This illustrates that none of the visual details are revealed if the correct key is not used. The image set covers both artificial ("Map" and "Document") and natural ("Lena" and "Tulips") types of images.

For reference, the number of bits in each of the bit classifications is given in Tables 3.2 to 3.5. As discussed in the previous section, the images show that in almost all cases, for the first coding iteration, almost no trees are partitioned during the LIS coding phase. This limits the expected variability of the decoder state after one coding iteration. Hence, it is advised that $K > 1$ to encrypt across two or more coding iterations to ensure sufficient variability of decoder state.

The Table 3.6 shows the total number of bits encrypted for different values of $K$, as well as the overall percentage of bits encrypted when the total bit-rate is 1 bpp. For $K = 2$, the percentage of encrypted bits is in the range of 0.2% to 0.4%. This extremely low, indicating a very low computational overhead to implement the encryption.
Finally, Figure 3.15 shows the coding performance of SecSPIHT compared to JPEG2000 and JPEG. SecSPIHT generally performs significantly better than JPEG. In most cases, SecSPIHT performs equal or better than JPEG2000. In lossless coding mode, SecSPIHT achieves the following bit-rates: i) “Lena”: 14.4 bpp (1.7:1 compression); ii) “Map”: 21.8 bpp (1.1:1 compression); iii) “Document”: 20.5 bpp (1.2:1 compression); and iv) “Tulip”: 10.7 bpp (2.2:1 compression). Clearly, like JPEG and JPEG2000, SecSPIHT performs better with natural image input, compared to artificial imagery.

3.5 Chapter Summary

This chapter presented the SecSPIHT secure image coder which achieves simultaneous compression and confidentiality of image data. Based on SPIHT, the SecSPIHT scheme performs code domain encryption on a small number of bits to achieve complete confidentiality of the entire image data. The selective encryption scheme does not negatively affect coding performance, resulting in coding performance consistently better than JPEG, and better than JPEG2000 in most cases. It also maintains the embedded scalability feature,
Figure 3.12: “Map” image and encrypted version (decrypted with incorrect key)

Figure 3.13: “Document” image and encrypted version (decrypted with incorrect key)
Figure 3.14: “Tulip” image and encrypted version (decrypted with incorrect key)

<table>
<thead>
<tr>
<th>Bit-plane</th>
<th>$B_{n,LIP}^{\text{sig}}$</th>
<th>$B_{n,LIP}^{\text{sgn}}$</th>
<th>$B_{n,LIS}^{\text{sig}}$</th>
<th>$B_{n,LIS}^{\text{sgn}}$</th>
<th>$B_{n,LIS}^{T\text{sig}}$</th>
<th>$B_{n,LSP}$</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_{\text{max}}$</td>
<td>256</td>
<td>30</td>
<td>0</td>
<td>0</td>
<td>256</td>
<td>0</td>
<td>542</td>
</tr>
<tr>
<td>$n_{\text{max}} - 1$</td>
<td>226</td>
<td>67</td>
<td>508</td>
<td>207</td>
<td>323</td>
<td>30</td>
<td>1361</td>
</tr>
<tr>
<td>$n_{\text{max}} - 2$</td>
<td>460</td>
<td>202</td>
<td>268</td>
<td>98</td>
<td>367</td>
<td>304</td>
<td>1699</td>
</tr>
<tr>
<td>$n_{\text{max}} - 3$</td>
<td>428</td>
<td>202</td>
<td>752</td>
<td>243</td>
<td>822</td>
<td>604</td>
<td>3051</td>
</tr>
<tr>
<td>$n_{\text{max}} - 4$</td>
<td>735</td>
<td>244</td>
<td>2428</td>
<td>741</td>
<td>2419</td>
<td>1049</td>
<td>7616</td>
</tr>
<tr>
<td>$n_{\text{max}} - 5$</td>
<td>2178</td>
<td>694</td>
<td>5288</td>
<td>1742</td>
<td>5192</td>
<td>2034</td>
<td>17128</td>
</tr>
</tbody>
</table>

Table 3.2: “Lena” – Number of bits in each classification for first 6 coding iterations
### Chapter 3. Secure Image Coder

#### Bit Classification

<table>
<thead>
<tr>
<th>Bit-plane</th>
<th>$B_{n,LIP-sig}$</th>
<th>$B_{n,LIP-sgn}$</th>
<th>$B_{n,LIS-sig}$</th>
<th>$B_{n,LIS-sgn}$</th>
<th>$B_{n,LIS-Tsig}$</th>
<th>$B_{n,LSP}$</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_{\text{max}}$</td>
<td>256</td>
<td>25</td>
<td>0</td>
<td>0</td>
<td>256</td>
<td>0</td>
<td>537</td>
</tr>
<tr>
<td>$n_{\text{max}} - 1$</td>
<td>231</td>
<td>86</td>
<td>132</td>
<td>36</td>
<td>284</td>
<td>25</td>
<td>794</td>
</tr>
<tr>
<td>$n_{\text{max}} - 2$</td>
<td>241</td>
<td>118</td>
<td>916</td>
<td>216</td>
<td>795</td>
<td>147</td>
<td>2433</td>
</tr>
<tr>
<td>$n_{\text{max}} - 3$</td>
<td>823</td>
<td>294</td>
<td>6016</td>
<td>1494</td>
<td>5016</td>
<td>481</td>
<td>14124</td>
</tr>
<tr>
<td>$n_{\text{max}} - 4$</td>
<td>5051</td>
<td>1634</td>
<td>13928</td>
<td>4288</td>
<td>12217</td>
<td>2269</td>
<td>39387</td>
</tr>
<tr>
<td>$n_{\text{max}} - 5$</td>
<td>13057</td>
<td>4443</td>
<td>54460</td>
<td>13062</td>
<td>43667</td>
<td>8191</td>
<td>136880</td>
</tr>
</tbody>
</table>

Table 3.3: “Map” – Number of bits in each classification for first 6 coding iterations

<table>
<thead>
<tr>
<th>Bit-plane</th>
<th>$B_{n,LIP-sig}$</th>
<th>$B_{n,LIP-sgn}$</th>
<th>$B_{n,LIS-sig}$</th>
<th>$B_{n,LIS-sgn}$</th>
<th>$B_{n,LIS-Tsig}$</th>
<th>$B_{n,LSP}$</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_{\text{max}}$</td>
<td>256</td>
<td>147</td>
<td>4</td>
<td>1</td>
<td>257</td>
<td>0</td>
<td>665</td>
</tr>
<tr>
<td>$n_{\text{max}} - 1$</td>
<td>112</td>
<td>15</td>
<td>408</td>
<td>95</td>
<td>470</td>
<td>148</td>
<td>1248</td>
</tr>
<tr>
<td>$n_{\text{max}} - 2$</td>
<td>410</td>
<td>156</td>
<td>884</td>
<td>279</td>
<td>892</td>
<td>258</td>
<td>2879</td>
</tr>
<tr>
<td>$n_{\text{max}} - 3$</td>
<td>859</td>
<td>292</td>
<td>4120</td>
<td>1139</td>
<td>3551</td>
<td>693</td>
<td>10654</td>
</tr>
<tr>
<td>$n_{\text{max}} - 4$</td>
<td>3548</td>
<td>1199</td>
<td>10120</td>
<td>3086</td>
<td>9117</td>
<td>2124</td>
<td>29194</td>
</tr>
<tr>
<td>$n_{\text{max}} - 5$</td>
<td>9383</td>
<td>3077</td>
<td>24340</td>
<td>7298</td>
<td>20085</td>
<td>6409</td>
<td>70592</td>
</tr>
</tbody>
</table>

Table 3.4: “Document” – Number of bits in each classification for first 6 coding iterations
Chapter 3. Secure Image Coder

### Bit Classification

<table>
<thead>
<tr>
<th>Bit-plane</th>
<th>$B_{n,LIP-sig}$</th>
<th>$B_{n,LIP-\text{sgn}}$</th>
<th>$B_{n,LIS-sig}$</th>
<th>$B_{n,LIS-\text{sgn}}$</th>
<th>$B_{n,LIS-Tsig}$</th>
<th>$B_{n,LSP}$</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_{\text{max}}$</td>
<td>256</td>
<td>112</td>
<td>0</td>
<td>0</td>
<td>256</td>
<td>0</td>
<td>624</td>
</tr>
<tr>
<td>$n_{\text{max}}-1$</td>
<td>144</td>
<td>103</td>
<td>28</td>
<td>5</td>
<td>266</td>
<td>112</td>
<td>658</td>
</tr>
<tr>
<td>$n_{\text{max}}-2$</td>
<td>64</td>
<td>24</td>
<td>352</td>
<td>121</td>
<td>354</td>
<td>220</td>
<td>1135</td>
</tr>
<tr>
<td>$n_{\text{max}}-3$</td>
<td>271</td>
<td>111</td>
<td>476</td>
<td>135</td>
<td>566</td>
<td>365</td>
<td>1924</td>
</tr>
<tr>
<td>$n_{\text{max}}-4$</td>
<td>501</td>
<td>199</td>
<td>956</td>
<td>335</td>
<td>1005</td>
<td>611</td>
<td>3607</td>
</tr>
<tr>
<td>$n_{\text{max}}-5$</td>
<td>923</td>
<td>368</td>
<td>1712</td>
<td>547</td>
<td>1923</td>
<td>1145</td>
<td>6618</td>
</tr>
</tbody>
</table>

Table 3.5: “Tulip” – Number of bits in each classification for first 6 coding iterations

### Test Image

<table>
<thead>
<tr>
<th>$K$</th>
<th>Lena</th>
<th>Map</th>
<th>Document</th>
<th>Tulip</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>256 (0.1%)</td>
<td>256 (0.1%)</td>
<td>260 (0.1%)</td>
<td>256 (0.1%)</td>
</tr>
<tr>
<td>2</td>
<td>990 (0.4%)</td>
<td>619 (0.2%)</td>
<td>780 (0.3%)</td>
<td>428 (0.2%)</td>
</tr>
<tr>
<td>3</td>
<td>1718 (0.7%)</td>
<td>1776 (0.7%)</td>
<td>2074 (0.8%)</td>
<td>844 (0.3%)</td>
</tr>
<tr>
<td>4</td>
<td>2898 (1.1%)</td>
<td>8615 (3.3%)</td>
<td>7053 (2.7%)</td>
<td>1591 (0.6%)</td>
</tr>
<tr>
<td>5</td>
<td>6061 (2.3%)</td>
<td>27594 (10.5%)</td>
<td>20721 (7.9%)</td>
<td>3048 (1.2%)</td>
</tr>
<tr>
<td>6</td>
<td>13527 (5.2%)</td>
<td>95111 (36.3%)</td>
<td>54444 (20.8%)</td>
<td>5683 (2.2%)</td>
</tr>
</tbody>
</table>

Table 3.6: The number of bits encrypted for the test images using different values of $K$. The percentage of bits encrypted for the test images coded at 1 bpp is shown in brackets.
Figure 3.15: Coding performance of the SecSPIHT secure coder, JPEG2000 and JPEG
allowing the code to be truncated to achieve arbitrary bit-rate.

An extensive analysis of the SPIHT code structure was provided in order to isolate the code bits that affect the execution path of the coder and decoder. The result of this was used in the design of SecSPIHT in the selection of the code bits to encrypt.

A security analysis was performed to determine the robustness of the scheme against attack. It was shown that SecSPIHT provides security equivalent to complete encryption with a secure cipher, if the security parameter was selected as $K > 1$.

Finally, simulation results were presented. Visually, the encrypted data was shown to reveal no image features. Setting the security parameter $K = 2$ requires only 0.2% to 0.4% of the overall data to be encrypted, with output bit-rate set at 1.0 bpp.
Chapter 4

Secure Visual Object-Based Coder for Privacy Protected Video Surveillance

This chapter presents a secure visual object-based coding scheme, called SecST-SPIHT. It combines the selective encryption principles of SecSPIHT, described in Chapter 3, with the ST-SPIHT object-based coding scheme (described in Section 2.3). SecST-SPIHT has the following key features:

- Complete confidentiality; arbitrarily-shaped object is completely obscured and can only be decoded and decrypted with the provision of a secret binary key

- Creates one bit-stream, coding both the shape and texture in one algorithm

- Encryption may be performed on the object texture, or both the object texture and shape

- Encryption is performed in the code domain — coding performance of ST-SPIHT is maintained
• Very computationally efficient; employs intelligent selective encryption which encrypts a very small portion of the code stream

• Bitstream maintains full scalability — it can be arbitrarily truncated to provide rate-distortion scalability of both the shape and texture

• Fully reversible; any loss is from coding (compression) operation, not from encryption

• Can be operated in a lossless fashion if integer colour transform and SA-DWT are utilized

SecST-SPIHT can be used as an enabling tool for privacy protection in video surveillance. Specifically, by applying it to private image regions in video surveillance, such as subjects’ face or body images, the data can be encrypted and protected from voyeurism and unauthorized use. If an incident occurs within the surveillance footage, a key may be used to decrypt the data when deemed necessary by an appropriate authority. This is in contrast to privacy protection schemes which irreversibly remove the private data. In these cases, it will generally be required to store the original in case an incident occurs and requires investigation, thus reducing the effective privacy measures. This chapter includes a privacy protection system description and discussion on the usage scenarios.

This chapter is organized as follows. Section 4.1 provides some important definitions and builds upon the SecSPIHT methodology to be applied to ST-SPIHT. Section 4.2 provides the complete SecST-SPIHT algorithm description. Section 4.3 presents a system for privacy protection in video surveillance based on the SecST-SPIHT secure object-based coder. Section 4.4 provides some simulation results on sample objects. The chapter finishes with a summary in Section 4.5.

It should be noted that no security analysis is presented in this chapter. The detailed security analysis on SecSPIHT presented in Chapter 3 applies to SecST-SPIHT as well.
4.1 Definitions and Methodology

This section will update the definitions presented in Section 3.1 as required to apply to the ST-SPIHT object-based coder. Note that the discussion will focus mainly on the unique aspects of ST-SPIHT to minimize repetition with the previous chapter. See Section 2.3 and [9, 10] for the full details of ST-SPIHT.

As with SPIHT, we denote the ST-SPIHT bit-stream as the ordered set of bits $B$. The bit-stream is divided into the ordered subsets $B = \{B_{n_{\text{max}}}, B_{n_{\text{max}}-1}, B_{n_{\text{max}}-2}, \ldots\}$, where $B_n$ is the set of bits obtained during coding iteration for bit-plane $n$ (i.e., representing the value $2^n$), and $n_{\text{max}}$ is the highest bit-plane at which coding is initiated. Each $B_n$ can be further subdivided into $B_n = \{B_{n,\text{LIP}}, B_{n,\text{LIS}}, B_{n,\text{LSP}}\}$, where $B_{n,\text{LIP}}$ denotes the ordered set of bits obtained during the first phase of the sorting pass where coefficients in the LIP are tested for significance; $B_{n,\text{LIS}}$ denotes the ordered set of bits obtained during the second phase of the sorting pass where entire trees are tested for significance; and $B_{n,\text{LSP}}$ denotes the ordered set of bits obtained during the refinement pass.

The ST-SPIHT scheme introduces the so-called “$\alpha$-test” functions which produce shape related code output directly the codestream. Accordingly, there are two additional bit classifications: i) $\alpha$-test shape bits, $B_{n,\text{LIP}}-\alpha$, for individual coefficients, produced during the LIP coding phase; and ii) $\alpha$-test shape bits, $B_{n,\text{LIS}}-\alpha$, for both individual coefficients and trees, produced during the LIS coding phase. This full updated decomposition of the bit-stream with example bits is shown in Figure 4.1. The interpretation of these bit classes is shown in Table 4.1.

It is clear from the ST-SPIHT coding algorithm (see Section 2.3) that the $\alpha$-test shape bits in both $B_{n,\text{LIP}}-\alpha$ and $B_{n,\text{LIS}}-\alpha$ affect the execution path of the decoder. They determine the subsequent state of the decoder by indicating whether the current coefficient is inside the object and should be coded, or is outside the object and should be ignored. Hence, we include these bits in the set of bits that requires encryption.
<table>
<thead>
<tr>
<th>Bit Type</th>
<th>Coding Phase</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B_{n,LIP-\alpha}$</td>
<td>LIP</td>
<td>$\alpha$-test shape bits for individual coefficients</td>
</tr>
<tr>
<td>$B_{n,LIP-sig}$</td>
<td>LIP</td>
<td>Significance bits for individual coefficients</td>
</tr>
<tr>
<td>$B_{n,LIP-sgn}$</td>
<td>LIP</td>
<td>Sign bits for individual coefficients</td>
</tr>
<tr>
<td>$B_{n,LIS-\alpha}$</td>
<td>LIS</td>
<td>$\alpha$-test shape bits for individual coefficients released during partitioning and whole trees</td>
</tr>
<tr>
<td>$B_{n,LIS-sig}$</td>
<td>LIS</td>
<td>Significance bits for individual coefficients released during partitioning</td>
</tr>
<tr>
<td>$B_{n,LIS-sgn}$</td>
<td>LIS</td>
<td>Sign bits for individual coefficients released during partitioning</td>
</tr>
<tr>
<td>$B_{n,LIS-Tsig}$</td>
<td>LIS</td>
<td>Significance bits for whole trees</td>
</tr>
<tr>
<td>$B_{n,LSP}$</td>
<td>LSP</td>
<td>Refinement bits</td>
</tr>
</tbody>
</table>

Table 4.1: ST-SPIHT bit classification
4.2 SecST-SPIHT Algorithm

The proposed SecST-SPIHT secure object-based coding system is shown in Figure 4.2. As with the SecSPIHT system, it adapts the non-secure coding system to include a secure coder, which utilizes a binary encryption key $k_E$ to produce a confidential bitstream. There is a matching secure decoding component which accepts a secret binary decryption key, $k_D$, which is required to correctly decode the object data.

The input object is defined by its shape and texture. The $M \times N$ full colour texture image is in the form

$$\mathbf{x} : \mathbb{Z}^2 \to \mathbb{Z}^3,$$

representing a two-dimensional matrix of three-component RGB (red, green, blue) colour samples

$$\mathbf{x}(i,j) = [x(i,j)_1, x(i,j)_2, x(i,j)_3],$$

with $i = 0, 1, ..., M - 1$ and $j = 0, 1, ..., N - 1$ denoting the spatial position of the pixel, and $x(i,j)_k$ denoting the component in the red ($k = 1$), green ($k = 2$), or blue ($k = 3$) colour channel.
The object shape is defined by a $M \times N$ binary shape mask, in the following form:

$$s : \mathbb{Z}^2 \rightarrow \{0, 1\},$$

representing a two-dimensional matrix of binary values with $i = 0, 1, \ldots, M - 1$ and $j = 0, 1, \ldots, N - 1$ denoting the spatial position and where $s(i, j) = 1$ denotes spatial positions “inside” the object, and $s(i, j) = 0$ denotes spatial positions “outside” the object.

As with SecSPIHT, for lossy coding, the ICT colour transform is utilized (see Section 2.2.1). In this case, the output texture pixel values will be real. For lossless coding, the RCT colour transform is used, producing integer output.

The SA-DWT utilizes the CDF 9/7 wavelet filters (see Appendix B) for lossy coding. For lossless coding, the Le Gall 5/3 integer filters are used. The output is the shape-adaptive transformed texture image, $x_T$.

The SecST-SPIHT coder is shown in more detail in Figure 4.3. The stream cipher encryption function $f_E(b, k_E)$ selectively encrypts individual bits $b$, using the key $k_E$. For SecST-SPIHT, the encrypted bits, $B_e$, are defined as:

$$B_e = \{B_{n, \text{LIP}-\text{sig}}, B_{n, \text{LIP}-\alpha}, B_{n, \text{LIS}-\text{sig}}, B_{n, \text{LIS}-\alpha}\}, \forall n > n_{\text{max}} - K \quad (4.4)$$

where $K$ is the input parameter controlling how many coding iterations during which encryption is performed. As with SecSPIHT, encryption of these bits makes the correct execution of the decoder impossible without first decrypting.

The SecST-SPIHT decoder is shown in Figure 4.4. It utilizes the decryption function, $f_E(b, k_D)$ which reverses the encryption operation of $f_E(b, k_E)$, using the secret decryption key $k_D$.

The complete coding algorithm is shown in Figure 4.5, and the Secure SCS subroutine in 4.6. We again use the controlled encryption function $f_cE(b, k_E, n, K)$, defined as follows:

$$f_cE(b, k_E, n, K) = \begin{cases} f_E(b, k_E), & n > n_{\text{max}} - K \\ b, & \text{otherwise} \end{cases} \quad (4.5)$$
Chapter 4. Secure Visual Object Coder

Figure 4.3: SecST-SPIHT secure object coder

Figure 4.4: SecST-SPIHT secure object decoder
Input: $x_T, s, \lambda, K, k_E$

1. **Initialization:** Find initial quantization level $n = n_{\text{max}} = \left\lceil \log_2 \left( \max_{(i,j,k)} \{|x_T(i,j,k)|\} \right) \right\rceil$; set LSP = $\emptyset$; set LIP = $\mathcal{H}$; set LIS = $\{(i,j)_k \text{ “type-A” } | (i,j)_k \in \mathcal{H}, \mathcal{D}(i,j)_k \neq \emptyset\}$.

2. **Sorting pass:**
   2.1. For each $(i,j)_k \in \mathcal{LIS}$:
      2.1.1. If $\alpha_p(i,j) = 1$ then:
          - Output $f_{E}(S_n(i,j)_k, k_E, n, K)$;
          - If $S_n(i,j)_k = 1$ then move $(i,j)_k$ to the LSP and output the sign of $x_T(i,j)_k$;
      2.1.2. If $\alpha_p(i,j) = 0$ then remove $(i,j)_k$ from the LIP;
   2.2. For each entry $(i,j)_k \in \mathcal{LIS}$:
      [If “type-A” entry, $\mathcal{T} = \mathcal{D}(i,j)_k$; If “type-B” entry, $\mathcal{T} = \mathcal{L}(i,j)_k$]
      2.2.1. If $n \geq \lambda$ and shape not completely coded, then:
          - If $\alpha_{SD}(\mathcal{T})$ not coded yet then output $f_{E}(\alpha_{SD}(\mathcal{T}), k_E, n, K)$;
          - If $\alpha_{SD}(\mathcal{T}) = 0$ then remove $(i,j)_k$ from the LIS and move on to next entry in the LIS (go to Step 2.2);
          - If $\alpha_{SD}(\mathcal{T}) = 1$ then:
              - If $\alpha_{SR}(\mathcal{T})$ not coded yet then output $f_{E}(\alpha_{SR}(\mathcal{T}), k_E, n, K)$;
              - If $\alpha_{SR}(\mathcal{T}) = 0$ and $n = \lambda$ then run SecSCS($\mathcal{T}$);
      2.2.2. If shape completely coded and $\alpha_{SD}(\mathcal{T}) = 0$ then remove $(i,j)_k$ from the LIS and move on to next entry in the LIS (go to Step 2.2);
      2.2.3. If “type-A” entry and $\alpha_{SD}(\mathcal{T}) = 1$:
          - Output $S_n(\mathcal{D}(i,j)_k)$;
          - If $S_n(\mathcal{D}(i,j)_k) = 1$ then:
              - For each $(p,q)_r \in \mathcal{O}(i,j)_k$:
                  - Output $f_{E}(S_n(p,q)_r, k_E, n, K)$;
                  - If $S_n(p,q)_r = 1$ then add $(p,q)_r$ to the LSP and output sign of $x_T(p,q)_r$;
                  - If $S_n(p,q)_r = 0$ and $\alpha_p(p,q)$ not coded yet, then output $f_{E}(\alpha_p(p,q), k_E, n, K)$;
                  - If $\alpha_p(p,q) = 1$ then add $(p,q)_r$ to the LIP;
              - If $\mathcal{L}(i,j)_k \neq \emptyset$ then move $(i,j)_k$ to the end of the LIS as “type-B” entry; else, remove $(i,j)_k$ from the LIS;
      2.2.4. If “type-B” entry and $\alpha_{SD}(\mathcal{T}) = 1$:
          - Output $S_n(\mathcal{L}(i,j)_k)$;
          - If $S_n(\mathcal{L}(i,j)_k) = 1$ then:
              - Add each $(p,q)_r \in \mathcal{O}(i,j)_k$ to the end of the LIS as “type-A” entry;
              - Remove $(i,j)_k$ from the LIS.

3. **Refinement pass:** For each $(i,j)_k \in \mathcal{LSP}$, except those found significant in the current sorting pass, output the $n^{th}$ most significant bit of $|x_T(i,j)_k|$;

4. **Quantization-step update:** Decrement $n$ by 1 and go to Step 2.

Figure 4.5: SecST-SPIHT coder algorithm
Chapter 4. Secure Visual Object Coder

Input: set $\mathcal{T}$ with root $(i, j)_k$, $n$, $k_E$, $K$

1. If $(i, j)_k$ is “type-A” entry:
   
   1.1. For each $(p, q)_r \in \mathcal{O}(i, j)_k$:
      
      1.1.1. If $\alpha_p(p, q)$ not coded yet then output $f_{cE}(\alpha_p(p, q), k_E, n, K)$;
      
      1.1.2. If $\mathcal{D}(p, q)_r \neq \emptyset$ then:
         
         • If $\alpha_{SD}(\mathcal{D}(p, q)_r)$ not coded yet then output $f_{cE}(\alpha_{SD}(\mathcal{D}(p, q)_r), k_E, n, K)$;
         
         • If $\alpha_{SD}(\mathcal{D}(p, q)_r) = 0$ terminate processing of $\mathcal{D}(p, q)_r$;
         
         • If $\alpha_{SD}(\mathcal{D}(p, q)_r) = 1$ then:
            
            – If $\alpha_{SR}(\mathcal{D}(p, q)_r)$ not coded yet then output $f_{cE}(\alpha_{SR}(\mathcal{D}(p, q)_r), k_E, n, K)$;
            
            – If $\alpha_{SD}(\mathcal{D}(p, q)_r) = 0$ then go to Step 1 treating $\mathcal{D}(p, q)_r$ as new “type-A” input;

   2. If $(i, j)_k$ is “type-B” entry:
      
      2.1. For each $(p, q)_r \in \mathcal{O}(i, j)_k$, go to Step 1 treating $\mathcal{D}(p, q)_r$ as new “type-A” input;

Figure 4.6: Secure Shape Code Set (SCS) Subroutine

This enables encryption for the first $K$ iterations and disables it after that.

Note that, as with ST-SPIHT, there is a parameter $\lambda$ which controls how many coding iterations after which the shape is forced to be coded completely. In other words, at the coding of bit-plane $\lambda$, the full shape is guaranteed to be losslessly coded. With small $\lambda$ (low bit-plane), the shape code will be spread across many coding iterations. With high $\lambda$ (up to bit-plane $n_{\text{max}}$), the shape coding will be completed early, most of it during the bit-plane $\lambda$.

The SecST-SPIHT coding operation can be terminated at any point based on a particular quality or bit-rate criterion. The SecST-SPIHT decoder is identical to the coder, except that bits are read from the stream rather than written, decrypted when necessary, and instead of performing significance tests, the SA-DWT reconstruction is updated accordingly based on the values read from the bit-stream.

In its default mode of operation, SecST-SPIHT makes the entire object confidential, meaning that both the object and the texture are protected. However, there are operating scenarios where the shape can be easily derived and therefore it makes little sense to encrypt it, or even encode it. For example, when segmenting a subject from background,
the shape of the object will be embedded as an outline in the background. If this data is available, then there is no need to code or encrypt it. In these cases, a flag is set to tell the coder that the shape has already been coded. Hence, when an $\alpha$-test is performed, the result of the test is simply not output to the bitstream. Visually, this means that if the secure object is attacked, the shape will be readily displayed, but the texture will be obscured. In fact, the shape must be made available for the decoder to be able to execute.

The next section describes how SecST-SPIHT may be deployed in a privacy protection system.

### 4.3 System for Privacy Protection in Video Surveillance

A video surveillance system is a system of video cameras and monitors intended for the surveillance of a physical space, such as a shopping mall, apartment building, or streetscape. At a minimum, the system must support the capture of imagery and the viewing of it to monitor the physical space. This is commonly called closed circuit television (CCTV) because the video signal is transmitted to designated monitoring stations, rather than broadcast. However, the communication channel might be wired or wireless, local or over a network. Most systems will also include some form of storage to allow footage to be automatically or manually stored for later retrieval. This may be used for investigative or evidentiary purposes after an event has occurred. In some cases, there is no “live” monitoring of the surveillance footage, thus necessitating storage to make use of the footage at a later time.

Figure 4.7 shows a generic video surveillance system. It consists of one or more cameras which are optionally connected (wired or wirelessly) to local monitors and/or storage. The data may also be transmitted over a network where there may be remote storage,
remote indexing/analytics, and remote monitors. In practice, surveillance systems may be configured into any number of diverse network topologies.

It’s clear that a surveillance system will often offer many points of access to the data. This is by design, since the intended purpose of video surveillance is that the footage be accessible for monitoring (either live or after the fact). This represents a significant problem with respect to privacy. The footage will typically contain a large quantity of data that would be considered to be private by many, including the facial and body images of private citizens, showing their activities and behaviours. A typical surveillance system does not offer much to protect this data. The protection measures may involve institutional policies on access and basic access control measures (such as computer passwords). However, since the footage must be readily available for the actual
surveillance purpose, the typical protection measures are considered by many to be wholly inadequate.

To design an effective privacy protection system for video surveillance, the following technical challenges must be addressed:

- Provide effective protection of private data from unauthorized use, across all possible access points

- The privacy protection system must not hinder the authorized surveillance activities by authorized personnel

- Any protection measures involving the removal of private visual data must be reversible in case the data is needed for investigative or evidentiary purposes by authorized personnel

Clearly, there must remain the concept of “authorized” use since the surveillance system is originally implemented for some purpose. A privacy protection system will rely on the owner of the surveillance system be a “friendly” institution, where the the surveillance system is not intended to invade privacy. In other words, the institution desires to protect the privacy of individuals and will have associated policies, but requires the technical means to implement them.

To address the stated technical challenges, a system for privacy protection in video surveillance is proposed, as illustrated in Figure 4.8. It utilizes SecST-SPIHT secure object coding as its core component for protection of private data. The system is a particular instance of the generic privacy protection system shown in Figure 2.22.

The system requires that private object data be defined, detected and tracked at the source. The mechanisms for this are beyond the scope of the discussion here, but in typical scenarios, this may include face tracking as a core component since the face visually represents the identity of subjects in video surveillance.

The system operates as follows:
Chapter 4. Secure Visual Object Coder

Figure 4.8: Proposed system for privacy protection in video surveillance
Chapter 4. Secure Visual Object Coder

1. During “normal” operation, private data such as faces, body images, license plates, etc., are detected, tracked, and segmented from the footage; the private data is defined in such a way that its removal would not hinder the intended surveillance purpose (i.e., faces may be removed to hide identity, but surveillance personnel can still monitor the activities of subjects).

2. The segmented private objects are coded and encrypted using SecST-SPIHT and a private key; the protected objects are stored for potential later use.

3. General security/surveillance personnel view the footage background with the private data having been removed; the image space in which the private data was originally present will be blurred, masked, or in-filled.

4. If an incident occurs and the protected data needs to be accessed for legitimate purposes, a process is initiated which requests the release of the key from an authorized person; with the key, the private data may be retrieved with no loss of fidelity\(^1\); the objects are composited with the background to reconstruct the original image frame.

The proposed approach, utilizing secure object-based coding, has a distinct advantage over other approaches, such as [62]. In [62], the scheme protects DCT blocks, which cannot exactly conform to arbitrary shapes. The proposed scheme utilizes a shape-adaptive transform and coding which offers pixel-level accurate, arbitrary shape definition. Additionally, in [62], the blocks have to remain encoded and stored with the rest of the frame. In contrast, the proposed object-based approach allows the private data to be completely segmented and securely stored separate from the background. This provides the flexibility to treat each object in an individualistic manner in terms of:

\(^1\)As previously noted in the discussion of SecSPIHT and SecST-SPIHT, any loss is from the compression routine, which is controllable. The encryption does not induce any loss in the data.
Coding Parameters: Each object is coded separately, can have a unique bit-rate and can be scaled individually

Security Parameters: Each object can be encrypted with a different key and different level of security

Indexing and Analytics: By storing each object separately, it is much simpler to analyze over a large number of frames

In this way, the proposed object-based approach offers superior flexibility compared to the other approaches in the literature.

Figure 4.9 shows a surveillance frame from an example implementation of the proposed privacy protection system. Figure 4.9(a) shows the original unprotected frame. Figure 4.9(b) shows the frame with the privacy protection measures in place. The private data (i.e., facial images in this case) has been removed and replaced with a blurred version. This footage still allows security personnel to monitor general activities. Figure 4.9(c) shows the frame with attempted access to the private data without the correct key. The secured objects reveal none of the original visual content of the object texture. Figure 4.9 shows the reconstructed frame, after the objects have been decoded and decrypted using the correct key. There is no loss of data in the objects, beyond compression loss.

To maximize privacy protection, the SecST-SPIHT protection for objects should be implemented as close to image capture as possible. This would reduce the number of possibly unprotected access points. The maximum protection would be provided by integrating SecST-SPIHT directly in the camera.

---

Footage acquired from TTC Bay Station in Toronto. Subjects appear voluntarily.
Figure 4.9: Example surveillance frames with privacy protection measures
4.4 Simulation Results

This section provides some simulation results, applying the SecST-SPIHT secure object coder to a selection of test objects. The SecST-SPIHT coder and decoder system were implemented in MATLAB and C code. All results utilized the lossy pipeline with the ICT colour transform (see Section 2.2.1), and the CDF 9/7 SA-DWT with 4 levels (see Section 2.3 and Appendix A).

Figures 4.10 to 4.13 show the test input. “Surveillance1” and “Surveillance2” were derived from actual surveillance footage; “Akiyo” and “Foreman” are standard MPEG test input. Two segmentation scenarios are considered: i) accurate shape; and ii) rough bounding box. In practice, detection and segmentation techniques may utilize the rough bounding box approach due to limitations on computational resources. For reference, the percentage of the frame occupied by the object is shown in Table 4.2.

Figures 4.14 to 4.21 show sample output using the test objects. In all cases, encryption
Figure 4.11: ‘Surveillance2’ test object.

Figure 4.12: ‘Akiyo’ test object.

Figure 4.13: ‘Foreman’ test object.
is performed during the first two coding iterations \((K = 2)\). In the cases where the shape is coded and encrypted with the object texture, the shape code is completed in the third iteration \((\lambda = n_{\text{max}} - 2)\). Figures 4.14 and 4.17 show the decrypted/decoded output ‘surveillance’ objects/frames when: (a)/(d) the correct decryption key is provided; (b)/(e) the incorrect decryption key is provided; and (c)/(f) the incorrect decryption key is provided, but the shape is available externally and only the texture is coded and encrypted. In all cases where the incorrect key is provided, the textural content is completely obscured; no object details can be seen. For the case (b)/(e) where the shape is coded and encrypted with the texture, the shape is also completely obscured. In order to reconstruct the frame without revealing the object shape mask, the background is transmitted as a full frame, with the missing texture information behind the object filled-in using prior frames.

Comparing the output of the accurately segmented objects with the bounding box segmented objects, it can be seen that the same level of obscuration is achieved when the shape is coded and encrypted (i.e., comparing Figures 4.14(e) with 4.15(e), and 4.16(e) with and 4.17(e)). However, in the cases where the shape has been provided externally, the accurate segmentation ((f) in Figures 4.14 and 4.16) may reveal silhouette details which could be used to identify subjects [77]. In contrast, the coarse bounding box ((f) in Figs. 4.15 and 4.17) completely obscures the actual shape of the object. The trade-off in this case is that the liberal nature of the bounding box segmentation map results in a large portion of the frame being obscured, reducing the ability to monitor general activities that occur in the frame.

Similarly, the decrypted/decoded test objects/frames ‘Akiyo’ and ‘Foreman’ (and rectangular bounding box versions) are shown in Figures 4.18 to 4.21, respectively with: (a)/(d) the correct decryption key provided; (b) the incorrect decryption key provided; and (c)/(e) the incorrect decryption key is provided, but the shape is available externally and only the texture is coded. In the cases when the shape is coded and encrypted with
the object and the incorrect decryption key is provided (Figures 4.18(b), 4.19(b), 4.20(b), and 4.21(b)), the full frame background is not transmitted since the prior frames in the sequence do not offer enough information to in-fill the original object area.

Figure 4.22 shows the fraction of the output code bits which are encrypted vs. the number of coding iterations during which encryption is performed ($K$). The total number of output code bits corresponds to a bit-rate of 2.4 bits-per-object-pixel (including the shape code for Figures 4.22(b) to 4.22(d)). Note that this higher bit-rate (compared to 1.0 bpp used for SecSPIHT in the previous chapter) is chosen because bit-rate is measured with respect as the ratio between file size and the number of object pixels (not frame pixels). With the small number of object pixels and the overhead of coding the shape, a higher bit-rate is selected to represent the scenario of high quality output. Figure 4.22(a) shows the case where the shape is not coded; Figures 4.22(b) to 4.22(d) show the cases where the shape code is completed during the first, second, and third coding iteration ($\lambda = n_{\text{max}}, n_{\text{max}} - 1, \text{and } n_{\text{max}} - 2$), respectively. In Figure 4.22(a), the effect of varying $K$ can clearly be seen, with the fraction of the output code being encrypted rising with $K$. The fraction remains small for all considered $K = 1, \ldots, 4$, ranging from approximately 0.2% to 1.6%. In Figures 4.22(b) to 4.22(d), a large jump in the portion of the bit-stream that is encrypted is observed once $K$ is set high enough to ensure that the shape is completely encrypted ($K = n_{\text{max}} - \lambda + 1$). When $K$ is raised above this point, the effect is more subtle since at low output bit-rates the shape code represents a significant portion of the bit-stream. With $K > n_{\text{max}} - \lambda$, the actual percentage of the output code that is encrypted is largely controlled by the portion which is the shape code ($B_{n_{\text{LIP}} - \alpha}$ and $B_{n_{\text{LIS}} - \alpha}$). If the user wishes to keep the level of encryption to a minimum for the purpose of computational efficiency, $\lambda$ should be set low enough to disperse the shape code further into the bit-stream, and setting $K \leq n_{\text{max}} - \lambda$ so that only the initial portion of the shape code is encrypted. In this case, $\lambda$ should be chosen so that $K$ can still be set high enough to encrypt a minimum number of bits to achieve a minimum
Figure 4.14: ‘Surveillance’ test object/frame decoded and decrypted output ($K = 2$): (a)/(d) with correct key; (b)/(e) with incorrect key; (c)/(f) with incorrect key and shape provided externally.
Figure 4.15: 'Surveillance1'-rect test object/frame decoded and decrypted output ($K = 2$): (a)/(d) with correct key; (b)/(e) with incorrect key and shape provided externally; (c)/(f) with incorrect key; (c)/ (f) with incorrect key and shape provided externally.
Figure 4.16: ‘Surveillance2’ test object/frame decoded and decrypted output ($K = 2$): (a)/(d) with correct key; (b)/(e) with incorrect key; (c)/(f) with incorrect key and shape provided externally.
Figure 4.17: Surveillance2-rect test object/frame decoded and decrypted output ($K = 2$): (a)/(d) with correct key; (b)/(e) with incorrect key; (c)/(f) with incorrect key and shape provided externally with incorrect key; (c)/(f) with incorrect key and shape provided externally.
Figure 4.18: ‘Akiyo’ test object/frame decoded and decrypted output \((K = 2)\): (a)/(d) with correct key; (b) with incorrect key; (c)/(e) with incorrect key and shape provided externally.
Chapter 4. Secure Visual Object Coder

Figure 4.19: 'Akiyo'-rect test object/frame decoded and decrypted output ($K=2$): (a)/(d) with correct key; (b) with incorrect key; (c)/(e) with incorrect key and shape provided externally.
Figure 4.20: ‘Foreman’ test object/frame decoded and decrypted output ($K = 2$): (a)/(d) with correct key; (b) with incorrect key; (c)/(e) with incorrect key and shape provided externally.
Figure 4.21: ‘Foreman’-rect test object/frame decoded and decrypted output ($K = 2$): (a)/(d) with correct key; (b) with incorrect key; (c)/(e) with incorrect key and shape provided externally.
desired level of security. For example, as in Figures 4.14 and 4.16, setting $K = 2$ and $\lambda = n_{\text{max}} - 2$ (i.e., shape code completed in the third coding iteration). The drawback of this approach is that the shape cannot be completely, losslessly decoded until later in the output bit-stream, possibly resulting in lossy shape reconstruction in very low bit-rate scenarios.

### 4.5 Chapter Summary

This chapter presented the SecST-SPIHT secure object-based coder. It retains all the properties of ST-SPIHT — efficient, combined coding of both object shape and texture — while adding complete confidentiality, requiring the encryption of only a small percentage of bits (ranging from 0.2% to 1.6%).

The SecST-SPIHT scheme is presented as part of a privacy protection solution for video surveillance. A privacy protection system was presented, which involved protecting private data objects (such as the face) using SecST-SPIHT. Since the encryption can be reversed with the provision of the correct key, the private data remains available if an incident occurs and the appropriate authority has released the key.
Figure 4.22: The fraction of bits encrypted vs. the security level parameter $K$ (number of encrypted coding iterations) for different $\lambda$ (shape code levels). The total bits in the code corresponds to a bit-rate of 2.4 bits-per-object-pixel.
Chapter 5

Conclusion

This thesis addressed the issue of securing visual data. Specifically, how to keep visual data confidential, via cryptographic techniques.

A secure wavelet-based image coder, called SecSPIHT was proposed. It codes and encrypts images efficiently, making the content confidential, controlled via a secret key. In most cases, the encryption operation is applied to less than 1% of the coded data, achieving full confidentiality while being very computationally efficient. Because the encryption is performed in the code domain, the compression performance is not negatively affected. Furthermore, all the desirable properties of the SPIHT coder are retained, including fine-grained rate-distortion scalability via truncation, and superior rate-distortion performance (compared to JPEG and JPEG2000). A full security analysis was provided. This indicated that the security parameter should be set $K > 1$ to ensure robustness against attack.

A related scheme, called SecST-SPIHT, was proposed for secure coding of arbitrarily-shaped visual objects. It codes and encrypts objects to enforce confidentiality. As with SecSPIHT, it requires that only a small number of code bits be encrypted. The scheme allows either the texture, or both the shape and texture to be protected.

Finally, a privacy protection system for video surveillance was proposed. It utilizes
SecST-SPIHT to protect private data objects, such as the face images of those appearing in surveillance footage. It allows these private data objects to be coded and encrypted soon after capture, thus preventing unauthorized access. Non-private data remains available for security personnel to effectively monitor the physical space. If an incident occurs in the surveillance footage that requires access to the private data (e.g., for investigation), an authorized person can release the key that allows the data to be decrypted. This is in contrast to irreversible methods (e.g., blurring and masking) which would require to have an unprotected copy of the data available in case of investigation.

5.1 Future Work

There are a number of areas touched upon in this thesis that provide direction for future research.

5.1.1 Addressing Error Control and Resiliency

The SecSPIHT and SecST-SPIHT approaches rely on the data dependent execution path of SPIHT and ST-SPIHT, respectively. If a code bit is altered, the execution path of the decoder is completely altered, resulting in a scrambling of the data after the location of the bit error. Clearly, these schemes do not tolerate errors (and, in fact, the proposed secure coding schemes exploit this feature).

In practical applications, error control and resiliency must be considered. First, the particular application scenario must be studied (i.e., the channel model) and the associated error source must be modelled. For example, wireless channels may be of interest in certain applications. Additionally, the application requirements must be defined with respect to what kind of output (visual) errors will be tolerated.

In MPEG video, error correction coding can be applied at the transport stream layer. However, the decoder will still tolerate the presence of errors (e.g., complete packet loss),
resulting in corrupted macroblocks, with propagation limited to the slice level.¹

SecSPIHT and SecST-SPIHT based systems can also benefit from error correction coding applied at the transport stream level. However, as previously noted, if an error is present that cannot be corrected, the entire remaining portion of the code will not be decodable. Because of the embedded nature of the codes, the result is in fact a reduced quality output (with the decoding of the initial portion of the code before the error). In some applications, this may actually be more desirable than the loss of a macroblock or slice. However, since none of the remaining code can be decoded, there will ultimately be a greater degree of data loss compared with MPEG. If this cannot be tolerated, then more robust error correction must be applied, with the trade-off of increased bandwidth and computational resources.

5.1.2 Addressing Inter-Frame Prediction in Video (Motion Compensation)

The proposed secure coding approaches deal with frames individually. That is, they do not consider motion compensation or inter-frame correlation. For MPEG-style video, this is not a great technical barrier, since most selective encryption schemes targeted to MPEG video will simply encrypt the motion vectors (they have been shown to reveal important details [4]). However, wavelet-based coding methods are not easily adapted to include motion compensation since the transform is usually applied on tiles larger than the macroblocks used for motion prediction. There are some works that address the issue of motion compensation applied to wavelet-based coding (e.g., [84]), however, this is still largely an open research area.

Alternatively, there are approaches that incorporate 3D wavelet transforms [36], to perform decorrelation in the third, temporal dimension. These approaches may be

¹Error correction coding may be applied at other layers of the communication system as well, depending on the network transmission protocol.
promising since SecSPIHT and SecST-SPIHT may be adapted to produce a single embedded code for a whole group of pictures (frames).

5.1.3 Stream Cipher Selection and Setup

This work assumed that a bit-level stream cipher would achieve perfect security over the encrypted portions of the data (i.e., equivalent to a one-time pad [40]). In reality, most ciphers have weaknesses, idiosyncracies, and particular operating requirements.

The design of the SecSPIHT and SecST-SPIHT can be more completely specified to include the selection of a particular cipher. In this case, the system should be analyzed so that the operating parameters of the stream cipher can be appropriately set to achieve the required security.

5.1.4 Alternative Cryptographic Approaches

SecSPIHT and SecST-SPIHT operate via direct, selective encryption of critical portions of the coded data. However, there may be opportunities to develop innovative schemes utilizing other cryptographic tools, such as secret sharing. It may be possible to apply such schemes to generate new functionality. For example, with secret sharing, it may be possible to split private data so that a “quorum” is required to decrypt it. While secret sharing has already been applied to images (e.g., [55]), it is not compatible with image and video coding. If such approaches are integrated into the secure coding approach, this may lead to novel methods with unique features.

5.1.5 Key Management Considerations

The proposed secure coding approaches assume that the secret keys are stored securely and are available when needed. This requires effective key management strategies. Additionally, the proposed system for privacy protected video surveillance supports multiple
keys, utilized for different objects. A complete system design requires the specification of a key management protocol which will protect the keys from unauthorized access but allow easy retrieval by those who are authorized.

5.1.6 Extension to Other Coding Approaches

While SPIHT has many desirable properties, it is not in common usage. JPEG and MPEG coders are the most common in deployed systems. It is logical to attempt to adapt the proposed approaches for these schemes. There are many technical barriers; SecSPIHT and SecST-SPIHT exploit some very unique properties of SPIHT. However, it would be desirable to attempt to generalize the concepts of code domain encryption so that they would be more widely applicable.
Appendix A

Discrete Wavelet Transform
Derivation and Properties

A.1 Orthogonal Wavelets

The DWT is in fact a class of transforms, with a particular transform defined by the choice of a “mother wavelet,” $\psi(t)$.$^1$ A particular mother wavelet, the Daubechies 4-tap, is shown in Figure A.1(a).

While our primary interest is in the discrete wavelet transform, its derivation requires reference to continuous-time wavelet functions. For a function to be a valid mother wavelet, it must have certain properties. The full list of properties can be found in [85, 22]; these properties include:

- Zero mean: $\int_{-\infty}^{\infty} \psi(t) dt = 0$

- Finite energy ($\psi(t) \in L^2(\mathbb{R})$): $\int_{-\infty}^{\infty} |\psi(t)|^2 dt < \infty$

- Compactly supported in the time-domain

$^1$Following convention, the wavelet functions are defined in the time domain ($t$). Spatial domain variables may be substituted with no loss of generality.
Appendix A. Discrete Wavelet Transform Derivation and Properties

121

(a) Mother Wavelet ($\psi(t)$)  
(b) Scaling Function ($\phi(t)$)

Figure A.1: Daubechies 4-tap wavelet

The result is that $\psi(t)$ has a band-pass frequency response.

A signal space can be defined by dilating and shifting (translating) $\psi(t)$ to create a set of basis functions as follows:

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}} \psi \left( \frac{t - b}{a} \right),$$  \hspace{1cm} (A.1)

where $a, b \in \mathbb{R}$. The parameter $a$ dilates and contracts the mother wavelet to affect the frequency response (i.e., frequency localization ability). The parameter $b$ shifts the mother wavelet to affect the temporal localization.

Of particular interest are the class of affine wavelets which support the discretization of $a$ and $b$ to produce a complete, orthogonal basis set for all $f(t) \in \mathbb{R}$. For these wavelets, the parameters are discretized as follows:

$$a = 2^{-m}, \quad b = n \cdot 2^{-m},$$  \hspace{1cm} (A.2)

where $m, n \in \mathbb{Z}$. The resultant basis functions are given as:

$$\psi_{m,n}(t) = 2^{m/2} \psi (2^m t - n).$$

As previously stated, this forms a complete, orthogonal basis set, where:

$$f(t) = \text{Span}_{m,n} \{ \psi_{m,n}(t) \}, \quad \forall f(t) \in \mathbb{R}.$$  \hspace{1cm} (A.3)
This is known as the dyadic wavelet basis.

For convenience, the mother wavelet is normalized to have unit $L^2$ norm ($||\psi(t)|| = 1$). Hence, the wavelet decomposition (forward transform) of a function $f(t)$ is performed as follows:

$$w(m,n) = \langle f(t), \psi_{m,n}(t) \rangle = \int_{-\infty}^{\infty} \psi_{m,n}(t)f(t)dt,$$

(A.4)

for $m, n \in \mathbb{Z}$. The corresponding signal reconstruction (reverse transform) is performed as follows:

$$f(t) = \sum_{m} \sum_{n} w(m,n)\psi_{m,n}(t).$$

(A.5)

When the goal is to perform a reversible decomposition (as it is in image coding), the decomposition described in Equation A.4 is impractical since the wavelet bases can only represent DC (zero frequency) content when $m \to -\infty$. To resolve this, a complementary function, known as the scaling function, $\phi(t)$, is introduced in conjunction with the mother wavelet. The scaling function corresponding to the Daubechies 4-tap mother wavelet is shown in Figure A.1(b). The properties of the scaling function can be found in [85]; a key property is that it has non-zero mean:

$$\int_{-\infty}^{\infty} \phi(t)dt \neq 0.$$  

(A.6)

As with the mother wavelet, it is typically normalized to have unit $L^2$ norm ($||\phi(t)|| = 1$). The complementarity of the scaling function with respect to the mother wavelet will become evident in the discussion that follows.

The scaling function is dilated and shifted in the same manner as the mother wavelet, as follows:

$$\phi_{m,n}(t) = 2^{m/2}\phi(2^mt - n),$$

(A.7)

for $m, n \in \mathbb{Z}$. A signal space is created from the scaling function as follows:

$$V_m = \text{Span}_n\{\phi_{m,n}(t)\},$$

(A.8)
where the fixed parameter \( m \) is considered the *resolution* or *scale* of the space. This gives rise to a series of nested subspaces, such that

\[ V_q \subset V_r, \quad \forall \ q < r. \quad (A.9) \]

\( V_q \) is said to be a lower resolution subspace of \( V_r \). The exact nature of this property of \( \phi_{m,n}(t) \) — known as the multiresolution property — is detailed later.

\( V_m \) serves as a resolution-based approximation space. The projection of a function \( f(t) \in \mathbb{R} \) onto this space produces the so-called “coarse” representation coefficients, \( c_m(n) \), calculated as follows:

\[ c_m(n) = (f(t), \phi_{m,n}(t)) = \int_{-\infty}^{\infty} f(t)\phi_{m,n}(t)dt. \quad (A.10) \]

The approximation of \( f(t) \) at resolution\(^2\) \( m \) is denoted as \( \phi_f^m(t) \), and is calculated via synthesis as follows:

\[ \phi_f^m(t) = \sum_n c_m(n)\phi_{m,n}(t). \quad (A.11) \]

We define a similar signal space at resolution \( m \) based on the mother wavelet, as follows:

\[ W_m = \text{Span}_n \{ \psi_{m,n}(t) \}. \quad (A.12) \]

\( W_m \) and \( V_m \) have the following important relationships:

- **Orthogonality:** \( V_m \perp W_m, \quad \forall \ m \in \mathbb{Z} \)

- **Completeness:** \( V_{m+1} = V_m \cup W_m \)

In other words, \( W_m \) is complementary to \( V_m \), representing the “details” lost when going from \( V_{m+1} \) to \( V_m \).

The projection of \( f(t) \) onto \( W_m \) produces the so-called “detail” coefficients, \( d_m(n) \), calculated as follows:

\[ d_m(n) = (f(t), \psi_{m,n}(t)) = \int_{-\infty}^{\infty} f(t)\psi_{m,n}(t)dt. \quad (A.13) \]

\(^2\)The term “resolution” is used loosely here as the nature of the “detail” that can be represented at resolution \( m \) is dependent on the particular choice of scaling function/mother wavelet.
Appendix A. Discrete Wavelet Transform Derivation and Properties

Note that detail coefficients $d_m(n)$ are equivalent to the wavelet coefficients $w(m, n)$ (Equation A.4) with fixed scale $m$. Based on the orthogonality and completeness properties and Equations A.10, A.11, and A.13, we get:

$$\phi_f^{m+1}(t) = \sum_n c_m(n) \phi_{m,n}(t) + \sum_n d_m(n) \psi_{m,n}(t),$$  \hspace{1cm} (A.14)

with $\phi_f^{m+1}(t) \in V_{m+1}$.

At this point, we seek to derive a discrete wavelet transform based on the continuous time theory discussed up to now. This is accomplished by way of the multiresolution property of the scaling function, expressed as follows: there exists a sequence $h(n)$ such that

$$\phi_{m,n=0}(t) = \sum_n h(n) \phi_{m+1,n}(t).$$  \hspace{1cm} (A.15)

This leads to what is known as the multiresolution analysis (MRA) equation:

$$\phi(t) = \sum_n h(n) \sqrt{2} \phi(2t - n).$$  \hspace{1cm} (A.16)

It can be seen clearly that the MRA gives rise to the nested subspace property described in A.9.

Based on the orthogonality and completeness properties of $V_m$ and $W_m$, an MRA equivalent property for the mother wavelet is given as follows: there exists a sequence $g(n)$ such that

$$\psi(t) = \sum_n g(n) \sqrt{2} \phi(2t - n).$$  \hspace{1cm} (A.17)

Using Equations A.10, A.13, A.16, and A.17, the following relationship between the coarse approximation and detail coefficients can be derived:

$$c_m(n) = \sum_k h(k - 2n)c_{m+1}(k)$$  \hspace{1cm} (A.18)

$$d_m(n) = \sum_k g(k - 2n)c_{m+1}(k)$$  \hspace{1cm} (A.19)

These very important results show that we can calculate the approximation and detail coefficients at one resolution level, $m$, from the approximation coefficients at the next
higher resolution level, \( m + 1 \). This is performed without having to refer back to the original signal \( f(t) \). Furthermore, the approximation coefficients \( c_m(n) \) are calculated by convolving (i.e., filtering) \( c_{m+1}(n) \) with the filter coefficients \( h(-n) \) and downsampling by a factor of two. Similarly, the detail coefficients \( d_m(n) \) are calculated by convolving \( c_{m+1}(n) \) with \( g(-n) \) and downsampling by two.

From Equations A.18 and A.19 the DWT is simply defined as follows: given a discrete-time signal \( x(n) \), consider this to be the coarse approximation, at arbitrary resolution \( M \), of a theoretical continuous time signal, so that:

\[
x(n) \rightarrow c_M(n).
\]  

Hence, the coarse approximation and detail coefficients at resolution \( m = M - L \), for any \( L \in \mathbb{N} \), are calculated from Equation A.20 and iterative application of Equations A.18 and A.19. The coefficients \( c_{M-L}(n) \) and \( d_{M-L}(n) \) are together considered the \( L \)-level DWT decomposition of \( x(n) \), requiring \( L \) iterations to calculate.

It should be noted that the transform coefficients \( c_m(n) \) and \( d_m(n) \) share the same independent discrete-time (or space) variable, \( n \), as the input. This is because the coefficients are ultimately calculated via discrete-time convolution (Equations A.18 and A.19). The coefficients in fact represent a subband decomposition of the input signal.

The filter coefficients \( h(-n) \) and \( g(-n) \) form low-pass \( (H') \) and high-pass \( (G') \) decomposition filters, respectively. Thus, a one-level discrete wavelet transform (DWT) involves filtering an input discrete time signal, \( x(n) \), using \( H' \) and \( G' \), and then downsampling by a factor of two to create the subbands \( x_{L1}(n) \) and \( x_{H1}(n) \), respectively (equivalent to \( c_{M-1}(n) \) and \( d_{M-1}(n) \), respectively — calculated from Equations A.18 and A.19). The next level DWT subbands, \( x_{L2}(n) \) and \( x_{H2}(n) \), can be obtained by repeating the procedure using \( x_{L1}(n) \) as the new input (i.e., iteration of Equations A.18 and A.19).

In practice, a particular DWT is defined via specification of \( h(n) \) and \( g(n) \), without requiring reference to the corresponding continuous time mother wavelet, \( \phi(t) \), and scaling function, \( \psi(t) \). Some important properties of \( h(n) \) and \( g(n) \) are provided below:
**Property 1:** Integrating both sides of Equation A.16:

\[
\int \phi(t)dt = \int_{-\infty}^{\infty} \sum_n h(n) \sqrt{2} \phi(2t - n)dt
\]

\[
= \sum_n h(n) \sqrt{2} \int_{-\infty}^{\infty} \phi(u) \frac{1}{2} du.
\]

Given that the scaling function has non-zero mean (Equation A.6), we get

\[
\sum_n h(n) = \sqrt{2}.
\] (A.21)

**Property 2:** Doing the same for the mother wavelet, using Equation A.17, and using the zero mean property of the mother wavelet, we get

\[
\sum_n g(n) = 0.
\] (A.22)

**Property 3:** Starting with the orthonormal property of the scaling function translates and using the Equation A.16 again,

\[
\delta(n) = \langle \phi(t), \phi(t - n) \rangle
\]

\[
= \int_{-\infty}^{\infty} \sum_j h(j) \sqrt{2} \phi(2t - j) \sum_k h(k) \sqrt{2} \phi(2(t - n) - k)dt
\]

\[
= \sum_j \sum_k h(j) h(k) \int_{-\infty}^{\infty} \sqrt{2} \phi(2t - j) \sqrt{2} \phi(2t - (k + 2n))dt
\]

\[
= \sum_j \sum_l h(j) h(l - 2n) \int_{-\infty}^{\infty} \sqrt{2} \phi(2t - j) \sqrt{2} \phi(2t - l)dt
\]

\[
= \sum_j \sum_l h(j) h(l - 2n) \langle \phi_{1,j}(t), \phi_{1,l}(t) \rangle
\]

\[
= \sum_j h(j) h(j - 2n)
\]

where the last step depends on the orthonormality of the translates of \( \phi_{1,n} \). We get:

\[
\langle h(n), h(n - 2k) \rangle = \delta(k).
\] (A.23)

**Property 4:** Using the orthonormal property of the mother wavelet translates and Equation A.17, the counterpart to Property 4 can be derived to get the final result:

\[
\langle g(n), g(n - 2k) \rangle = \delta(k).
\] (A.24)
**Property 5:** Using the orthogonality of the scaling function and mother wavelet translates, we get:

\[
0 = \langle \psi(t - n), \phi(t - p) \rangle \\
= \int_{-\infty}^{\infty} g(j)\sqrt{2}\phi(2(t - n) - j) \sum_k h(k)\sqrt{2}\phi(2(t - p) - k)dt \\
= \sum_j \sum_k g(j)\sqrt{2}\phi(2t - (j + 2n))\sqrt{2}\phi(2(t - (k + 2p)))dt \\
= \sum_l \sum_m g(l - 2n)h(m - 2p) \int_{-\infty}^{\infty} \sqrt{2}\phi(2t - l)\sqrt{2}\phi(2t - m)dt \\
= \sum_l \sum_m g(l - 2n)h(m - 2p) \langle \phi_{1,l}(t), \phi_{1,m}(t) \rangle \\
= \sum_l g(l - 2n)h(l - 2p)
\]

Finally, we get:

\[
\langle g(n - 2k), h(n - 2l) \rangle = 0, \quad \forall k, l . \quad (A.25)
\]

Summarizing the properties above, we find that the sum of the low-pass filter coefficients is \(\sqrt{2}\) and the sum of the high-pass filter coefficients is 0, as expected. Properties 3 to 5 tell us that \(h(n)\) and \(g(n)\) and their even translates are mutually orthonormal. Based on the relationship between the mother wavelet and the scaling function, an additional property exists as follows:

\[
g(n) = (-1)^n h(N - n) \quad (A.26)
\]

for filter length \(N\). This implies that the DWT is an orthonormal transform since it is implemented by filtering with \(h(-n)\) and \(g(-n)\) and downsampling by a factor of two. It should be noted that the properties were derived based on the parallel underlying definition of the mutually orthonormal scaling function and mother wavelet translates at a particular resolution. Since it is common to not have a closed form solution to the mother wavelet and scaling function, the sequences \(h(n)\) and \(g(n)\) are usually designed by starting with the perfect reconstruction quadrature mirror filter definition [18]. The
mutually orthonormal properties of the scaling function and mother wavelet can then be derived by reversing the derivations of Properties 3 to 5.

Given the above properties, the DWT definition is completed by using \( h(n) \) and \( g(n) \) to form the low-pass \((H)\) and high-pass \((G)\) reconstruction filters. The entire system of wavelet decomposition and reconstruction (or equivalently, analysis and synthesis) is shown in Figure 2.5. An arbitrary number of levels of decomposition may be performed up to the point where a subband signal length is equal to 1.

It should be noted that the linear convolution operation will produce an output that is longer than the input, via implicit zero-padding of the input signal. This is clearly undesirable for image coding applications (i.e., to have a greater number of wavelet coefficients than there are pixels in the original image). This may be resolved by instead employing a circular convolution (i.e., utilizing a periodic signal extension). However, this may result in artificial, high-frequency signal components (edges) which will reduce coding performance. The use of biorthogonal, symmetric wavelet filters allow for symmetric (mirror) signal extension to be used, avoiding the artificial edges. This class of wavelet filters is discussed in the next section. The details of the signal extension requirements are beyond the scope of the discussion here; the complete details are provided in [9].

### A.2 Biorthogonal Wavelets

It can be shown that it is not possible to have non-trivial symmetric FIR wavelet filters that are orthogonal [22, 86, 87]. Symmetric filters are guaranteed to have linear phase, which is a highly desirable feature in image compression since it allows for the preservation of edge locations within the subbands [88]. Furthermore symmetric filters allow for the use of symmetric extensions at the input signal boundaries [9]. Using symmetric extensions at the boundaries not only prevents signal expansion due to the zero-padding used in
the linear convolution, it also prevents unnatural sharp jumps at the boundaries which is common when periodic extensions are used.

In order to obtain symmetric, linear phase filters, the orthogonality condition is relaxed to produce *nearly* orthogonal filters with matching perfect reconstruction filters that form a biorthogonal set. As such, the impulse responses of the decomposition filters \( H' \) and \( G' \) are no longer \( h(-n) \) and \( g(-n) \) respectively. The underlying theoretical reconstruction basis functions \( \phi(t) \) and \( \psi(t) \) are no longer orthogonal and so wavelet filter Property 5 (Equation A.25) and Equation A.26 no longer hold.

The continuous time biorthogonal decomposition functions \( \phi'(t) \) and \( \psi'(t) \) are designed to generate the transform coefficients for the reconstruction bases \( \phi(t) \) and \( \psi(t) \), respectively. The MRA equations hold for the new scaling function and mother wavelet so that we have

\[
\phi'(t) = \sum_n h'(n) \sqrt{2} \phi'(2t - n) \tag{A.27}
\]

and

\[
\psi'(t) = \sum_n g'(n) \sqrt{2} \phi'(2t - n). \tag{A.28}
\]

The resulting relationship between the decomposition and reconstruction filters is as follows:

\[
g'(n) = (-1)^n h(N - n) \tag{A.29}
\]

and

\[
g(n) = (-1)^n h'(M - n) \tag{A.30}
\]

where \( N \) and \( M \) are the respective filter lengths. \( h'(-n) \) is the impulse response of the low-pass decomposition filter \( H' \), and \( g'(-n) \) is the impulse response of the high-pass decomposition filter \( G' \).

Property 3 (Equation A.23) and Property 4 (Equation A.24) are also no longer valid. Properties 3 to 5 are modified for biorthogonal wavelet filters as follows:
Biorthogonal Property 3:

\[ \langle h(n), h'(n - 2k) \rangle = \delta(k). \]  \hspace{1cm} (A.31)

Biorthogonal Property 4:

\[ \langle g(n), g'(n - 2k) \rangle = \delta(k). \]  \hspace{1cm} (A.32)

Biorthogonal Property 5:

\[ \langle g(n - 2k), h'(n - 2l) \rangle = 0, \quad \forall k, l \]  \hspace{1cm} (A.33)

and

\[ \langle g'(n - 2k), h(n - 2l) \rangle = 0, \quad \forall k, l . \]  \hspace{1cm} (A.34)

The Cohen-Daubechies-Feauveau (CDF) 9/7 biorthogonal wavelet filters (also known as the Daubechies 9/7 wavelet filters) [1] have been shown to provide excellent energy compaction for natural images. The CDF 9/7 DWT produces floating point output and is the transform utilized in JPEG2000 operating in lossy mode [16]. The integer output version of the Le Gall 5/3 DWT [89] is utilized for lossless coding. These are the two selected wavelet filters for the secure coding methods proposed in this thesis. They are implemented using the lifting framework, as described in Section 2.2.2. The integer modification (used for the Le Gall 5/3 DWT) is also described. The specific filter coefficients implementation steps for the two selected wavelet filters are given in Appendix B.
Appendix B

Selected Wavelet Filters

The two selected wavelet filters for the proposed secure image coding scheme are:

- Cohen-Daubechies-Feauveau (CDF) 9/7 filters which provide floating point output for lossy coding
- Le Gall 5/7 filters which provide integer output for lossless coding

These are the two wavelet filters used in the JPEG2000 standard for lossy and lossless coding, respectively [16, 18]. The following sections provide the implementations details.

B.1 Cohen-Daubechies-Feauveau 9/7 DWT (Floating Point)

The CDF 9/7 wavelet filters (also known as the Daubechies 9/7 wavelet filters) are biorthogonal, odd-length, symmetric [1]. The CDF 9/7 DWT produces floating point output and has been shown to provide excellent energy compaction for natural images. It is the transform utilized in the JPEG2000 standard [16] for lossy coding.

The lifting implementation is used in the proposed scheme (see Section 2.2.2); for the sake of completeness, the convolutional filter coefficients are shown in Table B.1 [1, 18].
Appendix B. Selected Wavelet Filters

132

Analysis Filter Coefficients

<table>
<thead>
<tr>
<th>n</th>
<th>$2^{-1/2} \cdot h'(n)$</th>
<th>$2^{-1/2} \cdot g'(n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.6029490182</td>
<td>0.5575435262</td>
</tr>
<tr>
<td>±1</td>
<td>0.2668641184</td>
<td>-0.2956358816</td>
</tr>
<tr>
<td>±2</td>
<td>-0.0782232665</td>
<td>-0.0287717631</td>
</tr>
<tr>
<td>±3</td>
<td>-0.0168641184</td>
<td>0.0456358816</td>
</tr>
<tr>
<td>±4</td>
<td>0.0267487574</td>
<td></td>
</tr>
</tbody>
</table>

Synthesis Filter Coefficients

<table>
<thead>
<tr>
<th>n</th>
<th>$2^{-1/2} \cdot h(n)$</th>
<th>$2^{-1/2} \cdot g(n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.5575435262</td>
<td>0.6029490182</td>
</tr>
<tr>
<td>±1</td>
<td>0.2956358816</td>
<td>-0.2668641184</td>
</tr>
<tr>
<td>±2</td>
<td>-0.0287717631</td>
<td>-0.0782232665</td>
</tr>
<tr>
<td>±3</td>
<td>-0.0456358816</td>
<td>0.0168641184</td>
</tr>
<tr>
<td>±4</td>
<td>0.0267487574</td>
<td></td>
</tr>
</tbody>
</table>

Table B.1: CDF 9/7 filter coefficients approximated to 10 decimal places [1]

The CDF 9/7 wavelet transform is performed using 4 main lifting steps and 2 normalization steps. The input signal is first extended using whole sample symmetric extensions to provide $x_{ext}(n)$ [9].

The steps to perform the transform are then:

\[
\begin{align*}
  y(2n + 1) & \leftarrow x_{ext}(2n + 1) + \lambda_1 \cdot (x_{ext}(2n) + x_{ext}(2n + 2)) \quad \text{(Step 1)} \\
  y(2n) & \leftarrow x_{ext}(2n) + \lambda_2 \cdot (y(2n - 1) + y(2n + 1)) \quad \text{(Step 2)} \\
  y(2n + 1) & \leftarrow y(2n + 1) + \lambda_3 \cdot (y(2n) + y(2n + 2)) \quad \text{(Step 3)} \\
  y(2n) & \leftarrow y(2n) + \lambda_4 \cdot (y(2n - 1) + y(2n + 1)) \quad \text{(Step 4)} \\
  y(2n + 1) & \leftarrow -K \cdot y(2n + 1) \quad \text{(Step 5)} \\
  y(2n) & \leftarrow 1/K \cdot y(2n) \quad \text{(Step 6)}
\end{align*}
\]

where “$\leftarrow$” is an assignment as part of the in-place operation which updates either the even or odd subsequence with the filtered version of the other [16, 90], and the lifting factors are defined as follows:

\[
\begin{align*}
  \lambda_1 &= -1.586134342, \quad \lambda_2 = -0.052980118, \\
  \lambda_3 &= 0.882911075, \quad \lambda_4 = 0.443506852, \\
  K &= 1.230174105
\end{align*}
\]

The output $y(n)$ is the interleaved subband sequence with the even subsequence $y(2n)$
Appendix B. Selected Wavelet Filters

being the low-pass output and the odd subsequence \( y(2n+1) \) being the high-pass output.

The reverse transform is performed by reversing the above steps. First, the interleaved subband sequence \( y(n) \) is extended using whole sample symmetric extensions to provide \( y_{ext}(n) \) [9]. The steps to reverse the transform are:

\[
\begin{align*}
    x(2n) & \leftarrow K \cdot y_{ext}(2n) & \text{(Step 1)} \\
    x(2n+1) & \leftarrow -1/K \cdot y_{ext}(2n+1) & \text{(Step 2)} \\
    x(2n) & \leftarrow x(2n) - \lambda_4 \cdot (x(2n-1) + x(2n+1)) & \text{(Step 3)} \\
    x(2n+1) & \leftarrow x(2n+1) - \lambda_3 \cdot (x(2n) + x(2n+2)) & \text{(Step 4)} \\
    x(2n) & \leftarrow x(2n) - \lambda_2 \cdot (x(2n-1) + x(2n+1)) & \text{(Step 5)} \\
    x(2n+1) & \leftarrow x(2n+1) - \lambda_1 \cdot (x(2n) + x(2n+2)) & \text{(Step 6)}
\end{align*}
\]

B.2 Le Gall 5/3 DWT (Integer)

For lossless coding, a DWT that maps inters-to-integers is required to guarantee exact invertibility. The Le Gall 5/3 integer wavelet transform is utilized in the JPEG2000 standard for lossless coding as it offers adequate energy compaction while being very computationally efficient. It can also be used for lossy coding, where computational efficiency is a primary requirement (e.g., for hardware implementations), at the cost of reduced coding performance.

The floating point transform which the 5/3 integer transform is based on has 5-tap and 3-tap low-pass and high-pass analysis biorthogonal filters, respectively [89]. This is modified to produce the integer version, according to framework described in Section 2.2.2.

As with the CDF 9/7 DWT, the input signal is first extended using whole sample symmetric extensions to provide \( x_{ext}(n) \) [9]. The 5/3 integer transform operation requires
only two lifting steps \[90,16,18,91\]:

\[
y(2n+1) = x_{\text{ext}}(2n+1) - \left\lfloor \frac{x_{\text{ext}}(2n) + x_{\text{ext}}(2n+2)}{2} \right\rfloor \tag{B.1}
\]

\[
y(2n) = x_{\text{ext}}(2n) + \left\lfloor \frac{y(2n-1) + y(2n+1) + 2}{4} \right\rfloor \tag{B.2}
\]

The output \(y(n)\) is the interleaved subband sequence with the even subsequence \(y(2n)\) being the low-pass output and the odd subsequence \(y(2n+1)\) being the high-pass output. Since the lifting coefficients are powers of 2, the implementation uses bit shifts instead of multiplication for highly efficient hardware implementation.

The 5/3 integer transform is easily reversed by tracing the lifting steps in reverse:

\[
x(2n) = y_{\text{ext}}(2n) - \left\lfloor \frac{y_{\text{ext}}(2n-1) + y_{\text{ext}}(2n+1) + 2}{4} \right\rfloor \tag{B.3}
\]

\[
x(2n+1) = y_{\text{ext}}(2n+1) + \left\lfloor \frac{x(2n) + x(2n+2)}{2} \right\rfloor \tag{B.4}
\]

where \(y_{\text{ext}}(n)\) is the interleaved subband sequence, extended using whole sample symmetric extensions \[9\].
Bibliography


