A thing is obvious mathematically after you see it.
R.D. Carmichael

University of Alberta

Partial Encryption for Image and Video Communication

by

Howard Chi Ho Cheng

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Computing Science

Edmonton, Alberta
Fall 1998

Abstract

The use of image and video communication has increased dramatically in recent years. When it is necessary to securely transmit data in limited bandwidth, both compression and encryption must be performed. Unfortunately, encryption and decryption are slow and so it is difficult to carry out real-time secure communication. Researchers have combined compression and encryption together to reduce the overall processing time [2, 3, 14, 23, 25], but most of them are insecure.

We propose an alternative solution, called *partial encryption*, in which a secure encryption algorithm is used to encrypt only part of the compressed data. Partial encryption is applied to several image and video compression algorithms in this thesis. Only 2%–27% of the compressed data is encrypted for typical images and video sequences, resulting in a significant reduction in encryption and decryption time. The proposed partial encryption schemes are fast, secure, and do not reduce the compression performance of the underlying compression algorithm.

Acknowledgements

I wish to thank my supervisor, Dr. Xiaobo Li, for his support in the duration of this work. Without his ideas, suggestions, criticisms, and friendship, it would not be possible to complete this work. In addition, this work was done as part of a project on image and video compression and encryption, and I have enjoyed working with other members of the group, especially Charles Cruden, Jason Knipe, Joseph Modayil, and Nicholas Roeder. Most importantly, they have often provided me with source code to different compression algorithms, saving me the effort to write the programs myself.

I wish to thank all my friends at the University of Alberta. I especially wish to thank Katerina Carastathis for her support and friendship for the last few years, and for her proofreading of this thesis. Joseph Modayil also offered valuable suggestions and advices on this work. Adam Beacham, Neil Burch, Alan Skelley, and Dr. Piotr Rudnicki of the programming team also made my year of graduate studies exciting and enjoyable. Of course, all the bridge players in the math lounge gave me a great opportunity to take a break from work.

I also wish to thank my family for their support throughout the years.

This work was supported by the Motorola Wireless Data Group and the Natural Sciences and Engineering Research Council under Grant OGP9198 and Postgraduate Scholarship (PGS A).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The use of image and video communication has increased dramatically in recent years. The World Wide Web and video conferencing are two examples. When communication bandwidth is limited, data is often compressed before transmission. If there is a need to protect the transmission from eavesdroppers, the transmission is also encrypted. For example, a wireless network often has limited bandwidth and its network traffic can easily be intercepted. As a result, transmissions over a wireless network need to be compressed and encrypted. Traditionally, an appropriate compression algorithm is applied to the multimedia data and its output is encrypted by an independent encryption algorithm. This process must then be reversed by the receiver.

Unfortunately, the processing time for encryption and decryption is a major bottleneck in real-time image and video communication. For example, a fast software implementation of the IDEA encryption algorithm [18, 19] given in [33] can encrypt or decrypt at a speed of 3.2 megabits per CPU second on a Pentium 150 processor running the Linux operating system. In addition, we must also take into account the processing time required for compression and decompression, for processing the associated audio data, and for other processing such as video capture and display, contrast adjustment, and so on. In the case of MPEG-1 compression, the bit rate may be up to 1.5 megabits per second, while MPEG-2 compression may produce output at a bit rate of 10 megabits per second or higher [26]. Compression algorithms for high-quality video sequences may give even higher bit rates. Encryption and decryption algorithms are too slow to handle the tremendous amount of data transmitted.

In many cases, the compression and decompression algorithms can barely keep up with the required bit rate even when they are accelerated by hardware. The additional processing required by encryption and decryption often makes it impossible to perform real-time secure video communication. The problem described above is also present in secure image and video storage. In this case, the "communication channel" is the storage device. The overhead for encryption and decryption may make real-time image and video processing infeasible. While hardware acceleration for encryption exists, software implementations are cheaper and more flexible. This is especially true for small, portable devices such as hand-held "videophones." Extra hardware may increase the cost of production, the size, and the power consumption of the device. A reduction in encryption and decryption time is important for these devices. Of course, a reduction in processing time is also important for more powerful computers.

Other researchers have also found inadequacies in the current state of the art in the area of secure image and video communication. Matias and Shamir [25] argued that standard cryptographic techniques are inadequate for video signals for the following reasons:

1. the transmitted signal is analog;

2. the transmission rate is very high;

3. the allowable bandwidth is limited.

While the first reason is no longer applicable as digital video has become feasible, the other two reasons remain valid. Both compression and encryption must be performed, and it is difficult to perform them quickly as recently noted by Chang and Liu [3].

They proposed an algorithm that combines image and video compression with encryption. The overhead of encryption and decryption is negligible in their algorithm, and they argue that it is fast and secure. Many researchers have examined the possibility of combining compression and encryption into a single process [2, 3, 14, 23, 25]. Inadequacies of these approaches are shown in this thesis.

In this thesis, we propose an alternative approach, called *partial encryption*, to reduce encryption and decryption time in image and video communication. In our proposed approach, only part of the compressed data is encrypted. This approach was first applied to quadtree image compression by the author in [4, 22]. In this thesis, we provide a more detailed analysis of the approach, introduce an application of partial encryption to wavelet compression algorithms, and extend our approach to video compression. In the next section, a general overview of our approach is given.

1.2 Partial Encryption Based On Decomposition

Image and video compression algorithms often decompose their input into logical parts. Some of these parts may be *important*, in the sense that they provide a significant amount of information about the original data. On the other hand, the remaining parts may not give much useful information by themselves. We propose that only the important parts are encrypted, and we call this approach *partial encryption*. Secure encryption algorithms such as IDEA [18, 19] can be used to encrypt the important parts. We use the term "information" loosely in this thesis, not referring to the concept of information defined mathematically by information theory. If a part can be used to reconstruct, approximate, or recognize the original data, we say that the part provides a significant amount of information.

When the size of the important parts is small compared to the size of the total output, a significant reduction in encryption and decryption time is achieved. As we have seen, encryption and decryption are bottlenecks in real-time secure image and video communication. Partial encryption may make real-time secure communication feasible. In particular, the processor may be idle when the communication channel is saturated, so that the important parts can be encrypted in parallel while the unimportant parts are transmitted. Thus, encryption can be carried out at a negligible cost. Decryption, on the other hand, may not be as easy to parallelize because the important parts are usually needed by the decompression algorithm before the unimportant parts. However, compression algorithms are often much slower than their corresponding decompression algorithms, so it is important to speed up the encoding process as much as possible.

In secure communication, a secret-key encryption algorithm is typically used to encrypt the transmission, and the secret key is encrypted by a public-key encryption algorithm such as the RSA algorithm [30]. The use of public-key encryption solves the problem of key exchange. However, public-key algorithms are too slow for encrypting a large amount of data. As a result, public-key algorithms are only used to encrypt the keys used in secret-key algorithms. If the important part is so small that public-key algorithms can be applied directly to the important part, secret-key encryption is unnecessary. The implementation and operating costs of secret-key encryption are completely eliminated.

We have examined several existing image and video compression algorithms, and found that those based on quadtrees [15, 22, 36, 37, 38, 39] and those based on zerotrees of wavelet coefficients [7, 15, 16, 32, 35] are suitable for partial encryption. In this thesis, we first examine image compression, and extend our results to video compression. It will be shown that the proposed schemes have several desirable properties:

Small important part. The important part in quadtree compression algorithms is 13–27% of the total output for all but one test images. The upper bound of the size of the important part is approximately 57%. The important part in the SPIHT algorithm based on zerotrees [32] is typically less than 2% of the total output for images of dimensions 512×512. Consequently, a significant reduction in encryption and decryption time is achieved. The important part in the SPIHT algorithm is so small that public-key encryption algorithms can be applied directly to the important part.

Simple implementation. Partial encryption can easily be applied to these two types of compression algorithms. The overhead in separating the important part from the unimportant part is negligible. It is also simple to combine the two parts together for decompression.

Uncompromised compression performance. The compression performance of the partial encryption schemes is the same as that of the original compression algorithms.

Security. There are no known attacks on these schemes. Furthermore, the schemes have properties that make them difficult to cryptanalyze.

1.3 Summary of Contribution

The contribution of this thesis includes:

Analysis of related algorithms. We examine a number of related algorithms that can be applied to secure image and video communication. We focus on the security provided by each proposed algorithm, and show that many of these algorithms are insecure.

A new view of data decomposition. Multimedia compression algorithms often decompose the input into multiple parts. The goal of many compression algorithms is to produce statistically uncorrelated parts. Each part can then be quantized independently, resulting in efficient compression. In addition to the criteria above, we also look at decomposition from an encryption perspective.

We examine the amount of information provided by each part, and argue that some parts need not be encrypted.

Application of partial encryption. The applicability of partial encryption to a number of existing image and video compression algorithms is examined. We examine the decomposition performed by these algorithms, and design partial encryption schemes for them. We study the security of each of our proposed schemes. Furthermore, analyses and experiments are performed to study the reduction in encryption and decryption time as well as the computational complexity of the proposed schemes .

The partial encryption schemes proposed in this thesis can be used in various applications that require image and video data to be compressed and encrypted before transmission. Real-time video conferencing and multimedia databases are examples of applications that may benefit from partial encryption.

1.4 Test Images and Video Sequences

Throughout this thesis, test images and video sequences are used to obtain experimental results. The test image set consists of 25 gray scale images that are frequently used by other researchers in image compression. For each image, the name, dimensions, and a short description are given in Table 1.1, and a few sample images are shown in Figure 1.1. Smooth images are those that contain large regions in which the intensity changes are gradual. Other images are usually dominated by fine details.

Two video sequences are used in the experiments. The two sequences are chosen so that one has relatively little motion, and the other one has a significant amount of motion. They are listed in Table 1.2, and the first frame of each sequence is shown in Figure 1.2. We only show the experimental results for the first ten frames of each sequence, as the remaining frames give similar results.

Table 1.1: Test images used in the experiments.

Image	Dimensions	Description
airfield	512×512	fine details
airplane	512×512	smooth
barbara	512×512	fine details
bay	256×256	smooth
bird	256×256	smooth
boat	512×512	smooth
bridge	256×256	fine details
camera	256×256	smooth
couple	512×512	smooth
crowd	512×512	fine details
festung	512×512	smooth
goldhill	512×512	fine details
io	512×512	fine details
lax	512×512	fine details
lena	512×512	smooth
man	512×512	smooth
mandrill	512×512	fine details
peppers	512×512	smooth
sailboat	512×512	smooth
shepherd	512×512	fine details, pencil drawing
sunset	256×256	smooth
washsat	512×512	fine details, low-contrast
woman1	512×512	smooth
woman2	512×512	smooth
zelda	512×512	smooth

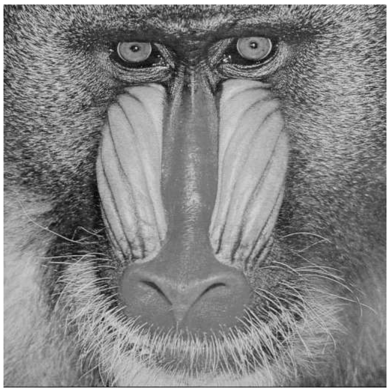
Table 1.2: Test video sequences used in the experiments.

Video Sequence	Dimensions	Description
claire	360×288	head-and-shoulder, little motion
football	720×496	sports, large amount of motion

(a) barbara

(b) camera

(c) lena



(d) mandrill

Figure 1.1: Example test images.

(a) claire

(b) football

Figure 1.2: First frame of each test video sequence.

1.5 Thesis Outline

The thesis is organized as follows. In Chapter 2, we define concepts and terminology commonly used in cryptography and compression. Then, in Chapter 3, we analyze related work by other researchers and demonstrate the inadequacies of these approaches. The proposed partial encryption approach is discussed in Chapter 4. This is followed by the application of partial encryption to image and video compression in Chapter 5 and Chapter 6, respectively. Finally, concluding remarks are given in Chapter 7.

Chapter 2

Background Information

In this chapter, we introduce concepts and terminology that are used in this thesis. It is by no means complete, and the reader is encouraged to consult [33] and [11] for more information on cryptography and image processing, respectively.

2.1 Cryptography

Cryptography is the study of keeping messages secure. When a message is to be transmitted in a hostile channel, the message can be encrypted to maintain privacy. The original message is called the *plaintext*, while the encrypted message is called the *ciphertext*. *Encryption* refers to the transformation from plaintext to ciphertext, and *decryption* refers to the inverse transformation. The encryption and decryption processes depend on a secret *key*, so that only those with the secret key can perform a successful decryption. The set of all possible keys in an encryption algorithm is called the *key space*.

Secret-key, or *symmetric*, algorithms use the same key for both encryption and decryption. On the other hand, *public-key*, or *asymmetric*, algorithms use different keys for encryption and decryption. The encryption key is easily derived from the decryption key, but it should be computationally infeasible to obtain the decryption key from the encryption key. The public encryption key is published and the messages encrypted with this key can be decrypted only by the corresponding decryption key. Public-key encryption algorithms eliminate the need of a secure channel for key exchange. However, they are much slower than secret-key algorithms when a large amount of data needs to be encrypted. Therefore, they are often used only to exchange the key used in secret-key algorithms.

Cryptanalysis is the study of breaking encryption algorithms. It is assumed that the cryptanalysts have full access to the description of the algorithms, as well as full access to the insecure channel in which a message is transmitted. There are a few types of attacks:

Ciphertext-only attack. The cryptanalyst has access to the ciphertext of several messages encrypted with the same key. The cryptanalyst attempts to recover the corresponding plaintext or the encryption key.

Known-plaintext attack. The cryptanalyst has access to the ciphertext and the corresponding plaintext for several messages encrypted with the same key. The cryptanalyst attempts to recover the key or to design an algorithm to decrypt any messages encrypted with the same key.

Chosen-plaintext attack. In this case, the cryptanalyst is allowed to choose the plaintext that is encrypted, and observe the corresponding ciphertext. The cryptanalyst's goal is the same as that in a known-plaintext attack.

Exhaustive key search. The cryptanalyst tests each of the possible keys one at a time until the correct plaintext is recognized. This attack is infeasible if the number of possible keys is at least 2^{128} [33]. This attack can be combined with any one of the three previous attacks to reduce the number of possible keys.

In practice, exhaustive key search and ciphertext-only attacks can usually be performed. Known-plaintext attacks can often be performed because many messages have known parts (eg. the header in a particular file format). Chosen-plaintext attacks may also be possible when the cryptanalyst has access to the encryption device.

Secure encryption algorithms should withstand all of the attacks described above provided that the key is not revealed.

2.2 Image and Video Processing

A *b-bit (gray scale) image* of dimensions $m \times n$ can be viewed as a two-dimensional function whose domain is the rectangular grid $\{0, \ldots, m-1\} \times \{0, \ldots, n-1\}$, and whose range type is the set of *gray levels*, or *intensities*, $\{0, \ldots, 2^b - 1\}$. Each pixel is indexed by its spatial coordinates, and is denoted by $f(r, c)$. We also call $f(r, c)$ the *pixel value* of the pixel at row r and column c. While this definition is made for gray scale images, a similar definition can be made for color images by changing the range type of f. A *video sequence* is simply a sequence of images, $\{f_i\}$, where i is an index to the frame in the sequence.

Two pixels at $p_1 = (r_1, c_1)$ and $p_2 = (r_2, c_2)$ are *4-neighbours* if and only if $(r_2, c_2) \in \{(r_1 \pm 1, c_1), (r_1, c_1 \pm 1)\}$, or equivalently, p_2 is the north, east, south, or west neighbour of p_1. The 4-distance between p_1 and p_2, denoted $D_4(p_1, p_2)$, is defined as the length of the shortest path from p_1 to p_2 such that two adjacent pixels in the path are 4-neighbours. Equivalently,

$$D_4(p_1, p_2) = |r_1 - r_2| + |c_1 - c_2|. \tag{2.1}$$

We define the *image histogram* of an image to be the frequencies of each of the gray levels in the image. In other words, it is a discrete function $h : \{0, \ldots, 2^b - 1\} \to \mathbb{N}$, where $h(k)$ is the number of times intensity k occurs in the image. The image histogram can be used to obtain information such as the average intensity of the image.

2.3 Quadtree

A *quadtree* is a rooted tree in which every node has 0 or 4 children, while a *4-ary tree* is a rooted tree in which every node has at most 4 children. Nodes with children are called *internal nodes*, while those without any children are called *leaf nodes*. Tree nodes having a common parent are called *sibling nodes*.

For each node in a tree, we define its *level* to be the number of edges in the shortest path from the node to the root. The *height* of the tree is defined to be the maximum of the levels of its nodes. Thus, a node at a low level is close to the root. It is important to observe that in the normal way of drawing trees, nodes at a high level are actually drawn below those at lower levels.

2.4 Image Compression

Image compression is the process by which an efficient representation is computed for an input image. *Decompression* is the process by which the image is reconstructed from its representation. In most cases, only the compression algorithm is described, as the decompression algorithm is a straightforward consequence.

Image compression algorithms can be classified into two categories. In *lossless* (or *reversible*) compression algorithms, the decoded image is the same as the original image. On the other hand, the decoded image in *lossy* (or *irreversible*) algorithms is only an approximation to the original. In general, lossy algorithms produce file sizes smaller than lossless algorithms. File sizes are generally measured in terms of *bits per pixel* (bpp), which is the average number of bits used to represent each pixel. The bits per pixel measure is also known as the *bit rate* of the algorithm. For a b-bit image, the uncompressed representation uses b bits per pixel for transmission and storage.

Most image compression algorithms published in the literature compress only gray scale images. It is usually straightforward to apply an algorithm to color images by separating a color image into three gray scale component images. As a result, we will only study algorithms that compress gray scale images. In addition, many compression algorithms compress only images of dimensions of $2^n \times 2^n$ for some integer $n \geq 0$. If the input image does not have the required dimensions, pixels having the same gray level can be added to "extend" the image. Since the extended pixels form a large homogeneous region, the compression performance is not significantly affected. We will always assume that images have the appropriate dimensions.

2.4.1 Quadtree Image Compression

Many variations of quadtree image compression algorithms exist [15, 22, 36, 37, 38, 39]. We only describe the basic concept here. Although more powerful compression algorithms exist, the computational complexity of quadtree compression is low and it performs better than the algorithm by the Joint Photographic Experts Group (JPEG) [29] at low bit rates [15].

In the lossless version, the algorithm starts with a tree with one node. If the entire image is homogeneous, the root node is made a leaf and the gray level describing the entire image is attached to the leaf. Otherwise, the image is partitioned into four quadrants and four corresponding children are added to the root of the tree. The algorithm then recursively examines each quadrant, using each of the four children as the root of a new subtree. This process must terminate as quadrants of dimensions 1×1 are homogeneous. Thus, the image is represented by a quadtree and the gray levels at the leaf nodes. An image can be compressed if there are large homogeneous regions. Lossless quadtree compression works best on computer graphics or line art with large homogeneous regions. It can also be used on medical images in which any information loss is unacceptable.

The lossy version is similar to the lossless version, but the test for homogeneity of a square block is replaced by a test for similarity. The similarity of the pixels in a block can be measured by the variance of the pixel values, texture information, and other kinds of statistics. The values attached to the leaf nodes are parameters that describe the block. Some examples of the parameters are average gray level and parameters for a first-order model [38]. Lossy quadtree compression works best on photographic images.

Quadtree compression can be implemented in either a *top-down* or a *bottom-up* fashion. The description given above corresponds to a top-down implementation. In a bottom-up implementation, the algorithm starts with a complete quadtree that corresponds to a decomposition consisting of only individual pixels. The highest level of the quadtree is first examined, and four sibling leaf nodes are merged if they are homogeneous or similar. This is repeated at the next highest level until there are no leaf nodes at a level, or if the root is reached. In practice, a bottom-up implementation is often preferred because it is more efficient.

2.4.2 Wavelet Image Compression

In this section, we describe the basic concept of wavelet transforms. This is followed by a description of the Set Partitioning in Hierarchical Trees algorithm [32], which is a member of a class of wavelet image compression algorithms based on zerotrees [7, 15, 16, 35]. Details are given to facilitate the discussion of the proposed partial encryption schemes.

We first introduce the one-dimensional wavelet transform, and extend it to the two-dimensional wavelet transform. Given a one-dimensional input sequence $C^{(0)}$, the *wavelet transform* of $C^{(0)}$ can be viewed as the application of a pair of filters H and G to $C^{(0)}$ and the filtered output. The filter H is a low-pass filter and the filter G is a high-pass filter. The frequency ranges of the output sequences are at most half of that of the input sequence, so that the filtered output can be subsampled by a factor of two. Two output sequences $C^{(1)}$ and $D^{(1)}$ are produced from $C^{(0)}$ by H and G, respectively. The process is then recursively applied to $C^{(1)}$. In general, if $*$ denotes the convolution operator and S denotes the subsampling operator, we have

$$C^{(i+1)} = S\left(H * C^{(i)}\right) \tag{2.2}$$

$$D^{(i+1)} = S\left(G * C^{(i)}\right), \tag{2.3}$$

where the recursion stops when the length of $C^{(i)}$ is less than some threshold. Fur-

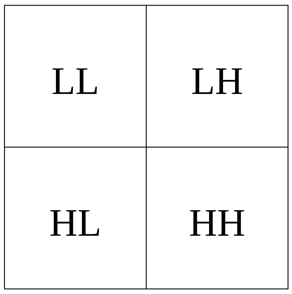


Figure 2.1: The four output bands of the wavelet filters.

thermore, there exists a pair of filters H^* and G^* such that

$$C^{(i)} = H^* * U\left(C^{(i+1)}\right) + G^* * U\left(D^{(i+1)}\right), \tag{2.4}$$

where U is an upsampling operator which increases the number of samples by a factor of two. As a result, the original sequence $C^{(0)}$ can be reconstructed from the sequences $C^{(l)}, D^{(l)}, D^{(l-1)}, \ldots, D^{(1)}$, where l is the number of times the filters are recursively applied. The values in the sequences $C^{(l)}, D^{(l)}, D^{(l-1)}, \ldots, D^{(1)}$ are called the *wavelet coefficients* of $C^{(0)}$.

The two-dimensional wavelet transform of an image can be performed by the one-dimensional transform on the columns, followed by the one-dimensional transform on the rows. Each application of the filters creates four bands in the image, labelled LL, LH, HL, HH in Figure 2.1. The first letter in the label corresponds to the frequency range (low or high) in the vertical direction, and the second letter corresponds to the frequency range in the horizontal direction. The label is sometimes called the *orientation* of the coefficient band, as it indicates the direction of the features represented by the band. The transform is recursively applied to the LL band until its dimensions are smaller than some threshold, resulting in a hierarchy of bands, sometimes called a *pyramid decomposition*, as shown in Figure 2.2. The number in the band label is the *pyramid level* of the band. The inverse transform is carried out in the reverse order of each operation.

For most natural images, the wavelet coefficients in the high-frequency bands are usually small. In addition, there is often a correlation between coefficients that are

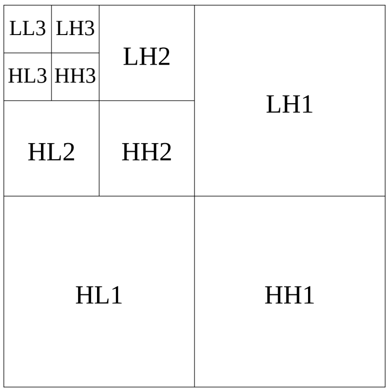


Figure 2.2: The hierarchy of wavelet coefficient bands.

in different pyramid levels of the hierarchy. For example, the coefficients in HL2 are correlated to those in HL3. We now describe the Set Partitioning in Hierarchical Trees (SPIHT) algorithm [32], which takes advantage of these properties to improve compression performance. We follow the notation and development of the algorithm as presented by Said and Pearlman in [32].

We define the only LL band to be the *root level*. For each 2×2 block of coefficients in the root level, three trees of coefficients are defined such that each coefficient outside the root level has four children in the band with the same orientation, but at a lower pyramid level. This is shown in Figure 2.3. Note that the trees are grown in the opposite direction of the pyramid levels. We denote the coefficient at (i, j) as $c_{i,j}$. The following sets of coefficients are defined:

- $\mathcal{O}(i, j)$: the set of coordinates of the children of the coefficient at (i, j);
- $\mathcal{D}(i, j)$: the set of coordinates of all descendants of the coefficient at (i, j);
- $\mathcal{H}$: the set of coordinates of all coefficients in the root level;
- $\mathcal{L}(i, j) = \mathcal{D}(i, j) - \mathcal{O}(i, j)$.

Given a threshold $T = 2^n$, a set of coefficients S is *significant* if there is a coefficient

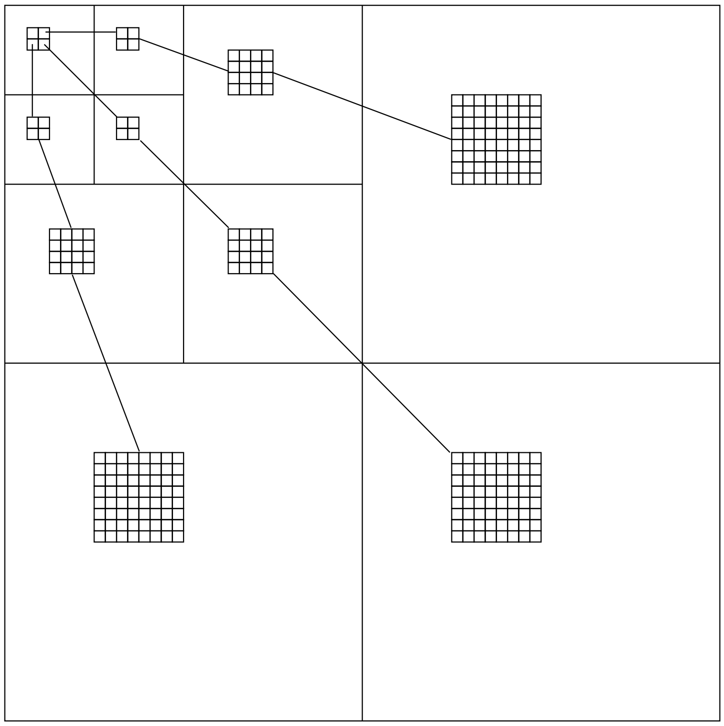


Figure 2.3: The trees of wavelet coefficients.

in S whose magnitude is at least T. We define the function

$$S_n(S) = \begin{cases} 1 & \text{if } S \text{ is significant with respect to } T = 2^n, \\ 0 & \text{otherwise.} \end{cases}$$

For convenience, $S_n(\{(i,j)\})$ is simply denoted as $S_n(i,j)$. Three lists are maintained by the algorithm: the list of insignificant sets (LIS), the list of insignificant pixels (LIP), and the list of significant pixels (LSP). The LIS contains two types of entries. Type A entries represent the sets $\mathcal{D}(i,j)$ and type B entries represent the sets $\mathcal{L}(i,j)$. The LIP is a list of insignificant coefficients that do not belong to any of the sets in the LIS. The LSP is a list of coefficients that have been identified as significant.

The SPIHT algorithm encodes the wavelet coefficients by selecting a threshold such that $T \leq \max_{(i,j)} |c_{i,j}| < 2T$, where (i,j) ranges over all coordinates in the coefficient matrix. The significant coefficients are identified and moved to the LSP from the LIS or the LIP. All coefficients in LSP that have been identified as significant in previous passes are then refined in a way similar to binary search. The threshold is decreased by a factor of two, and the above steps are repeated. The encoding process stops when the desired bit rate is reached. The output is fully *embedded*, so that the output at a higher bit rate contains the output at all lower bit rates embedded at the beginning of the data stream. As a result, the decoder may stop receiving bits at any point and decode the image. The compression performance of the algorithm depends mainly on its ability to represent large sets of insignificant coefficients with only a few bits. The term *zerotree* refers to a set of insignificant coefficients grouped into a tree.

The algorithm described by Said and Pearlman is now presented [32]:

1. **Wavelet transform:** perform wavelet transform on image so that the root level has dimensions 8×8.

2. **Initialization:** output $n = \left\lfloor \log_2 \left(\max_{(i,j)} |c_{i,j}| \right) \right\rfloor$; initialize the LSP as an empty list, and add the coordinates $(i,j) \in \mathcal{H}$ to the LIP, and only those with descendants also to the LIS, as type A entries.

3. **Sorting pass:**

   (a) for each entry (i,j) in the LIP do:

      i. output $S_n(i,j)$;

      ii. if $S_n(i,j) = 1$ then move (i,j) to the LSP and output the sign of $c_{i,j}$;

   (b) for each entry (i,j) in the LIS do:

      i. if the entry is of type A then

      - output $S_n(\mathcal{D}(i,j))$;
      - if $S_n(\mathcal{D}(i,j)) = 1$ then
        - for each $(k,l) \in \mathcal{O}(i,j)$ do:
          * output $S_n(k,l)$;
          * if $S_n(k,l) = 1$ then add (k,l) to the LSP and output the sign of $c_{k,l}$;

  * if $S_n(k,l)=0$ then add (k,l) to the end of the LIP;

- if $\mathcal{L}(i,j) \neq \phi$ then move (i,j) to the end of the LIS, as an entry of type B; otherwise, remove entry (i,j) from the LIS;

ii. if the entry is of type B then

- 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 an entry of type A;
  - remove (i,j) from the LIS.

4. **Refinement pass:** for each entry (i,j) in the LSP, except those added in the last sorting pass, output the n-th most significant bit of $|c_{i,j}|$;

5. **Quantization-step update:** decrement n by 1 and go to Step 3.

2.5 Video Compression

Video compression algorithms take advantage of the correlation among successive frames to compress video sequences. These algorithms generally consist of two parts—motion compensation and residual error coding. In *motion compensation*, a frame is segmented into objects or blocks. The movement, or *motion vector*, of each object or block between successive frames are transmitted. The decoder uses the previous frame and the motion vectors to predict the current frame. However, some prediction error may exist. The *residual error* of the prediction is the difference between the current frame and the predicted frame. The residual error must be transmitted as well.

The first frame of a video sequence is compressed by an image compression algorithm, and subsequent frames are encoded by the motion vectors and the residual errors. The prediction by the motion vectors is not accurate if there are dramatic changes from one frame to the next, such as a change of scene. In addition, the quality of the prediction tends to degrade over time. Thus, video compression algorithms usually treat frames at fixed intervals as *intraframes* and encode them without any prediction. The algorithm effectively restarts the encoding at every intraframe. In this way, degradation in the quality of the image frames do not accumulate.

Motion compensation is typically performed by segmenting an image frame into blocks and determining the motion vector for each block. Exhaustive search in a small neighbourhood is used to find a block in the previous reconstructed frame that matches best with the current block. Because of the exhaustive search, the encoding process is slow. On the other hand, the decoder only has to perform the prediction specified by the given motion vectors, and so it is much faster than the encoder.

If the prediction by motion compensation is accurate, the magnitudes of residual error are small. As a result, the residual error has low entropy and tends to be highly compressible. The residual error can be treated as an image, so that any image compression algorithm can be used to encode the residual error.

Chapter 3

Analysis of Related Work

In this chapter, we examine related work by other researchers in this area. There have been a few proposals on the combination of image compression and encryption, and these proposals are often insecure. In this chapter, we summarize each proposal and point out its shortcomings. Unless otherwise stated, the attacks presented are original work of this thesis.

3.1 Concealing Adaptive Model in Huffman and Arithmetic Coding

Huffman coding [11, 12] and arithmetic coding [11, 41] are two lossless algorithms for compressing arbitrary data. Both algorithms use statistics of the input data in the encoding process. In particular, the probabilities of the input symbols are used by many implementations of these algorithms. If the statistics are not known, we may collect the statistics in one pass and perform the encoding in the second pass. The statistics must be stored with the compressed file for decoding, and the overhead in storing the statistics may offset the space reduction obtained by compression.

Adaptive data modelling can be used to solve the above problem. Instead of using a static probability model of the input, statistics are gathered and updated as symbols are coded. Both the encoder and decoder start with the same initial model. Each symbol is encoded using the statistics gathered from the beginning of the data stream up to, but not including the current symbol being encoded. After a symbol is encoded, the encoder updates its probability model. On the other hand, the decoder decodes each symbol using only the statistics gathered so far, and updates its probability model based on the symbol decoded. As long as both the encoder and the decoder start with the same initial model, the compression can be reversed.

Jones [14] proposed algorithms that use modified splay trees to maintain the probability models for Huffman coding and arithmetic coding. Splay trees are self-balancing binary search trees that are adjusted after every search. When a symbol is encoded in Huffman coding, the path from the root of the tree to the symbol is modified by "semi-rotations" about the nodes on the path, so that the path length is halved. If a symbol is encountered many times, the path length to that symbol, and hence its code length, becomes small. A similar representation can be used to store symbol frequencies needed by an arithmetic coder.

In adaptive Huffman coding and arithmetic coding, errors in the first few bits of the encoded data may destroy the synchronization between the encoder and the decoder, making the remaining data unrecoverable. Based on this observation, Jones suggested that the adaptive compression algorithm can also be used as an encryption algorithm if the initial model is kept secret. The encryption key is a string of symbols that is encoded before the encoding or decoding of the actual message. Thus, the initial model is updated based on the symbols in the encryption key. On Jones' World Wide Web home page [13], it is stated that this scheme is vulnerable to chosen-plaintext attacks. According to Jones, there are sequences of input symbols that can expose the current state of the model. This security of the algorithm could be strengthened by inserting random data periodically, where the period must be sufficiently small to break the chosen plaintext into short segments [13]. However, the insertion of random data may reduce the compression performance of the algorithm.

3.2 Combined Arithmetic Coding and Encryption

In arithmetic coding, the entire message is represented by some interval of real numbers. To each source symbol, we assign a subinterval whose length is proportional to its probability in the source. When a symbol is coded, we choose the appropriate interval and subdivide the chosen interval. In an adaptive algorithm, the probabilities are updated after a symbol is coded. A probable symbol does not reduce the current interval as much as a less probable symbol. As a result, if many probable symbols are coded, the final interval is large. Since large intervals can be coded with relatively few bits, data compression is achieved.

Liu *et al.* [23] recently proposed an algorithm that incorporates encryption into arithmetic coding. Unlike the scheme described in Section 3.1, this scheme does more than simply making the adaptive model secret. Its main feature is that when an interval is chosen for the encoded symbol, small secret adjustments are made to the interval. The low end of the interval is increased by multiplying by a secret multiplier of the form 1.0***, and the high end is decreased by multiplying by a secret multiplier of the form 0.9***. In fact, two pairs of secret multipliers are available, and the choice of the pair depends on a separate bit string.

The authors claim that the algorithm has many desirable properties for security, such as diffusion and the randomness of output. No cryptanalysis is known for the algorithm at this time. However, their experimental results show that the algorithm is approximately twice as slow as the original arithmetic coding algorithm by Witten *et al.* [41]. In addition, the compression performance is 2% worse than that of the original algorithm.

3.3 Video Scrambling With Space-Filling Curves

Matias and Shamir [25] proposed a video encryption algorithm in which each frame in the video is scrambled by a secret space-filling curve. A space-filling curve is a

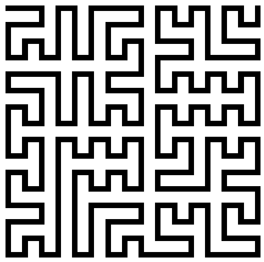


Figure 3.1: A space-filling curve on an image of dimensions 16×16.

continuous curve that visits every pixel exactly once. An example of a space-filling curve is shown in Figure 3.1. A space-filling curve specifies a bijective mapping from a two-dimensional image to a one-dimensional array such that two consecutive pixels in the array are 4-neighbours in the original image. As a result, much of the spatial correlation in the original image is preserved, and algorithms are available for compressing images scanned by space-filling curves [27, 28, 40]. Furthermore, there are approximately $2^{0.5573n}$ space-filling curves available as secret keys for an $n \times n$ image [25], making exhaustive key search infeasible. The authors suggested that the space-filling curve should be changed occasionally for successive frames.

3.3.1 A Ciphertext-Only Attack

Using correlation among successive frames, Bertilsson *et al.* [1] were able to cryptanalyze the encryption algorithm, recovering many pieces of the original image frame. Suppose there is little or no motion in several consecutive frames. If a different space-filling curve is used for every frame, the same image portions are scanned by many different space-filling curves. A cryptanalyst can take advantage of the information obtained from multiple scans.

The number of pixels between two pixels x and y in the scanned array is an upper bound on $D_4(x, y)$ in the original image. Let up_1, up_2 be unique pairs of adjacent pixels appearing in the scanned arrays, and $n_i(up_1, up_2)$ be the number of pixels between the two pairs in the scanned array of the i-th frame. Then, the maximum possible distance between the two unique pairs in the original image, $d_{\max}(up_1, up_2)$, is

$$d_{\max}(up_1, up_2) = \min_i n_i(up_1, up_2). \tag{3.1}$$

Most false unique pairs that are not unique in the original image frame can be detected if many scans are available [1]. Once unique pairs are identified, some of the pixels around each unique pair may also be recovered, giving many small pieces of the frame. Using $d_{\max}(up_1, up_2)$ on the unique pairs within the pieces, the maximum possible distance between two pieces p_1 and p_2, $D_{\max}(p_1, p_2)$, is

$$D_{\max}(p_1, p_2) = \min_{\substack{up_1 \in p_1 \\ up_2 \in p_2}} d_{\max}(up_1, up_2). \tag{3.2}$$

Pieces that have small distances among each other can be joined or put close together. In this way, large pieces of the image can be recovered. Consequently, Bertilsson *et al.* [1] concluded that space-filling curves are not suitable for encrypting sequences of images.

3.3.2 Known-Plaintext and Chosen-Plaintext Attacks

The scrambling scheme can be viewed as a transposition cipher that uses space-filling curves to specify the permutation applied to the plaintext. Consequently, it is vulnerable to known-plaintext and chosen-plaintext attacks.

In a known-plaintext attack, the scrambling effect on pixels with unique gray levels can be determined. As a result, the number of possible space-filling curves that can be used to encrypt the image is restricted. This technique can be extended to study the effects of unique pairs to further reduce the number of possible space-filling curves. In general, the number of possible space-filling curves decreases as more plaintext-ciphertext pairs are available.

In a chosen-plaintext attack, we construct input images with many unique pixels. If each image is of dimensions $n \times n$ and has m unique pixels, then n^2/m encryptions are needed to completely determine the space-filling curve. For example, an

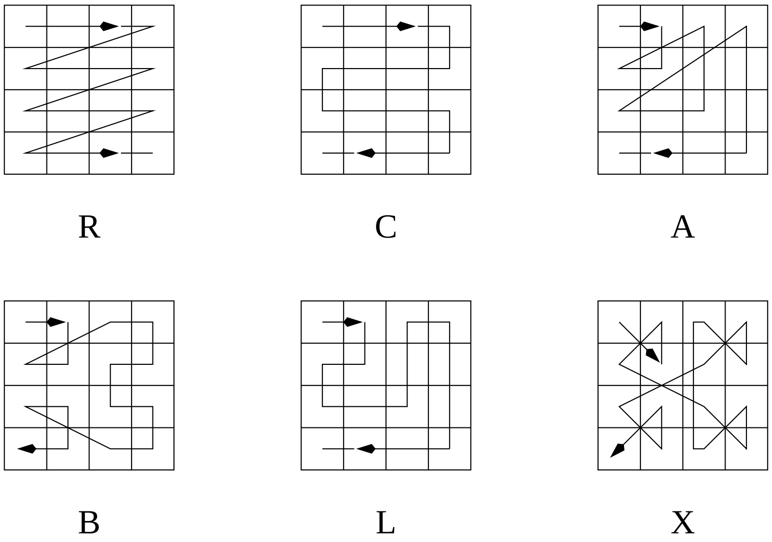


Figure 3.2: Some SCAN letters and their scan patterns on 4×4 images.

8-bit image of dimensions 512×512 has at most 255 unique pixels. In this case, only $512^2/255 \approx 1028$ encryptions are required. The space-filling curve encryption algorithm is insecure against the chosen-plaintext attack.

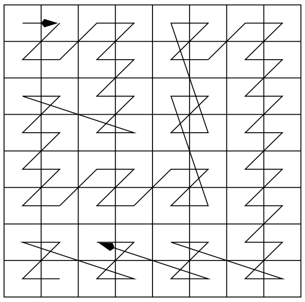
3.4 Image Scrambling With Hierarchical Scan Patterns

Bourbakis and Alexopoulos [2] proposed another image encryption algorithm that scrambles an image using hierarchical scan patterns defined by the SCAN language. The SCAN language is a context-free language in which every string is of the form

$$w_t = L_1 n_1 \# L_2 n_2 \# \cdots \# L_t n_t,$$

where L_i is a symbol in the alphabet set, n_i is a power of 2, and t is the length of the string. For an image of dimensions $n \times n$, it is required that $\Pi_{i=1}^{t} n_i = n$.

The alphabet set contains 15 letters that describe scan orders defined in the language. Examples of the scan orders for 4×4 images are shown in Figure 3.2. A string w_t in the SCAN language defines a hierarchical scan pattern. The image is first decomposed into n_1^2 square blocks of dimensions $(n/n_1) \times (n/n_1)$, ordered by the scan pattern specified by L_1. Then, each block is decomposed into $n_2 \times n_2$ blocks



1	2	1	2	1	2	1	2
2	2	2	2	2	2	2	2
1	2	1	2	1	2	1	2
2	2	2	2	2	2	2	2
1	2	1	2	1	2	1	2
2	2	2	2	2	2	2	2
1	2	1	2	1	2	1	2
2	2	2	2	2	2	2	2

Figure 3.3: The SCAN pattern defined by $L4\#R2$ on an 8×8 image. Also shown is the lowest level at which each pixel becomes a drive point.

ordered by the scan pattern corresponding to L_2. This process is repeated t times and the last decomposition results in blocks of dimensions 1×1. We define drive points to be pixels at which scan patterns begin. For convenience, we define a trivial scan pattern on 1×1 blocks, so that every pixel in the image becomes a drive point after some levels of decomposition. Once a pixel becomes a drive point, it remains a drive point at higher levels of decomposition. An example of a scan pattern and the lowest level at which each pixel becomes a drive point is shown in Figure 3.3.

We now describe the SCAN image encryption algorithm. Given a b-bit image of size $n \times n$, the encryption key is a pair (w_t, v) where w_t is a string from the SCAN language, and $v = (v_1, v_2, \ldots, v_t) \in \{0, 1, \ldots, 2^b - 1\}^t$ is an additive noise vector. The encryption is carried out by scanning the image with the specified SCAN pattern, followed by substitutions of the pixel values defined by

$$E(p) = \left(p + \sum_{i=k(p)}^{t} v_i\right) \mod 2^b, \tag{3.3}$$

where $k(p)$ is the lowest level at which p becomes a drive point. An image is decrypted by putting the scanned pixels into a two-dimensional array and reversing the substitutions.

Bourbakis and Alexopoulos also suggested that the algorithm can be used as an image compression algorithm by representing homogeneous image regions by their sizes and intensities. One possibility is to search for homogeneous regions during the

Table 3.1: Number of distinct SCAN patterns for an $n \times n$ image [2].

n	Number of distinct SCAN patterns
2	15
4	237
8	3753
16	59433
32	1022475
64	1.562473×10^7
128	2.599637×10^8
256	4.027959×10^{10}
512	6.617867×10^{10}
1024	1.036039×10^{12}
2048	1.687737×10^{13}

decomposition process—if an image block is homogeneous, no further decomposition is performed on that block. As shown in Figure 3.3, SCAN patterns are not necessarily continuous, and large image regions may be decomposed into many small blocks. This may reduce the compression performance. Unfortunately, the authors discussed this as part of their future research, and no experimental results were given.

3.4.1 Cryptanalysis

Bourbakis and Alexopoulos stated that "the power of the SCAN encryption scheme is mainly based on its transposition part" [2]. The number of distinct SCAN patterns is also stated in [2] and is presented in Table 3.1. Even for $n = 2048$, the number of distinct SCAN patterns is only 1.687737×10^{13}, which is less than 2^{44}. Without the substitutions, the encryption algorithm is vulnerable to exhaustive key search [33]. In addition, the SCAN encryption scheme is a transposition cipher without the substitutions, and the same known-plaintext and chosen-plaintext attacks for space-filling curve encryption can be used.

The substitutions performed on the image pixels are image-independent. That is, the same substitution is performed on each pixel location regardless of the image content. We can take advantage of this property to remove the substitutions in a chosen-plaintext attack. We construct an input image in which every pixel has intensity 0. The encrypted output, E_0, of this image gives the effect of the substitutions on the entire image. For every image encrypted with the same key, we simply subtract each pixel in the scanned array by the corresponding pixel in E_0. The effect of the substitutions is completely removed, and the encryption algorithm can be attacked by exhaustive key search, known-plaintext attacks, and chosen-plaintext attacks. In addition, the substitution on each pixel depends only on the lowest level at which the pixel becomes a drive point. Consequently, we may discover t and n_i as well.

The authors suggested that more complicated substitution schemes can be used in place of the one proposed. However, the security of the SCAN encryption algorithm would depend more on the substitution part, which contradicts the authors' intention that security should depend mainly on the transposition part. The authors did not provide an analysis on the relative importance of the two parts of the algorithm, and it is not clear if a stronger substitution scheme will make the resulting algorithm secure.

3.5 Quadtree Image Compression and Encryption

Chang and Liu [3] recently proposed an image compression and encryption algorithm based on lossless quadtree image compression. When a square block is partitioned into four quadrants, four children are added to an internal node in some order. Each ordered child corresponds to a quadrant in the partition, and the correspondence determines the scanning order in which the quadrants are examined. If the order is different at every internal node, then it may be difficult to recover the image without knowing the orders used at each internal node. This is the basis of Chang and Liu's quadtree image compression and encryption algorithm. For an image of dimensions $2^n \times 2^n$, the encryption key is a "scan word" of the form $W = W_0 W_1 \ldots W_{n-1}$, where W_i specifies the scanning orders for the nodes at level i. Each W_i has the

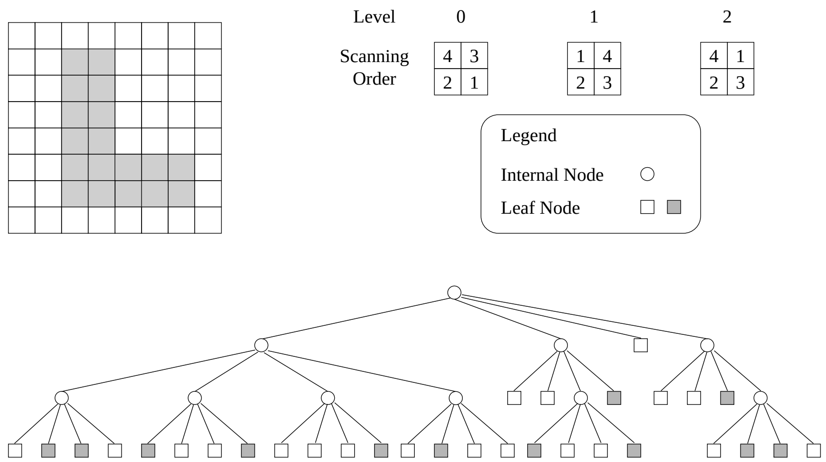


Figure 3.4: The effect of quadtree image compression and encryption on a binary image.

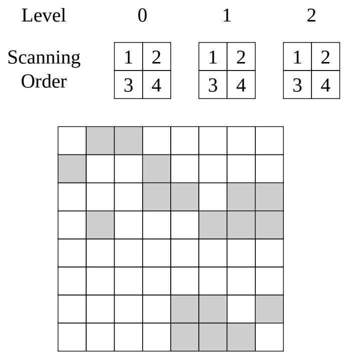


Figure 3.5: Reconstructed image from the quadtree in Figure 3.4 using a fixed scanning order.

form $W_i = L_1 L_2 \ldots L_t$, in which $t = 4^i$ and each L_j denotes one of the $4! = 24$ scanning orders of the quadrants. The key completely specifies the order in which the four quadrants are to be examined at each internal node. The operation of the compression and encryption algorithm is illustrated in Figure 3.4. For simplicity, the same scanning order is used for each node at the same level. The scrambling effect of the encryption algorithm is shown in Figure 3.5, in which we naively decode the image using the same scanning order for every internal node.

From the experimental results shown in [3], it is clear that the compression performance of the algorithm is poor for photographic images (only 1% reduction in file size). In fact, the only image in the test set that is compressed significantly is a binary image with large homogeneous regions. Furthermore, the length of the key is at least $\lceil \log_2(24)(4^n - 1)/3 \rceil$ bits, which is exponential in n. Although the compression performance can be improved using a lossy quadtree compression algorithm [15, 22, 36, 37, 38, 39], the exponential key length makes the algorithm unsuitable for practical use.

Chang and Liu argued that the number of possible keys is large, and exhaustive search is computationally infeasible. Furthermore, they argued that the original image cannot be recovered without the encryption key. We now show that the effective key space can be significantly reduced for many images, and that some information about the image can be obtained without the encryption key. We will also show that the algorithm is vulnerable to known-plaintext and chosen-plaintext attacks. As a result, this algorithm should not be used in transmitting images over an insecure channel.

3.5.1 Key Space Reduction

For an image of dimensions $2^n \times 2^n$, Chang and Liu [3] showed that the number of possible keys is $24^n \times 4^{n(n-1)/2}$. However, the calculation presented was incorrect as the number of choices on independent decisions were added, not multiplied. The number of keys should be $24^{(4^n-1)/3}$, which is much larger than that stated by the authors.

Although the number of keys is extremely large, the number of effective keys can be reduced dramatically. That is, different keys may give the same ciphertext on a particular image. Suppose an internal node has four children that are not all distinct, so that there are at least two identical quadrants. Since there is repetition in the objects being permuted, the number of scanning orders is reduced by a factor of at least two. In the worst case, all four quadrants are identical and there is no effective scanning order at that internal node.

There are other kinds of scanning orders that are not suitable for use. For example, many scanning orders at the root node simply rotate or reflect the image. The number of these kinds of scanning orders at other nodes depends on the scanning orders used by ancestor nodes. We will not attempt to determine the number of these kinds of scanning orders.

It should be noted that the scanning order specified for a node is applied only if the node is an internal node. If significant compression on a particular image is expected, many nodes in the complete quadtree will not be examined in the algorithm—their ancestors are leaf nodes. In particular, most of the nodes corresponding to small block sizes will not be examined. In a complete quadtree, the number of nodes at each level grows exponentially, so that the majority of the nodes in a complete quadtree are not examined. It follows that a more accurate upper bound on the number of effective keys is 24^m, where m is the number of internal nodes in the quadtree. It is difficult to make general comments about this quantity as m depends on the input image.

Although the effective key space is reduced significantly, exhaustive search still appears to be computationally prohibitive. With today's technology, exhaustive search is computationally infeasible if $m \geq 28$ [33], and most nontrivial images have more than 28 internal nodes in their quadtree representations. Next, we examine an attack to reduce the search space even further.

3.5.2 An Attack Based on Image Histogram

The histogram of an encrypted image can be recovered without the encryption key. Since the leaf nodes in the encrypted representation have exactly the same intensities and sizes as those in the unencrypted representation, the number of pixels having each of the possible intensity levels can be calculated. This observation is demonstrated in Figures 3.4 and 3.5.

The image histogram can be used in an attack in the following manner. The average intensity of an image represents a very low-resolution approximation of the original image, and is easily calculated from the histogram. Assuming that the scanning order of the root node is known, we can obtain the histograms of each of the four quadrants; hence an image with twice the resolution in each dimension can be obtained. By applying this technique repeatedly, we can obtain an approximation of resolution $2^k \times 2^k$ using the average intensities of the blocks at level k, provided that the scanning orders used at levels $0, 1, \ldots, k-1$ are known. A reasonable approximation can be obtained with $k = 5$ in most cases. Although the number of internal nodes with level ≤ 4 can still be large (at most $(4^5 - 1)/3 = 341$), the search space is reduced significantly. For example, if the image size is 512×512, then the upper bound on the size of the search space is reduced from 24^{87381} to 24^{341}. Further reduction may be obtained when combined with the key space reduction by ineffective scanning orders. Unfortunately, exhaustive search may still be infeasible.

3.5.3 A Known-Plaintext Attack

One undesirable property of the quadtree image encryption algorithm is that pixels in the same quadrant in the original image are also in the same quadrant in the encrypted image, as shown in Figures 3.4 and 3.5. This property can be used in a known-plaintext attack to reduce the number of possible keys, making an exhaustive search more efficient.

Given an encrypted image as well as the corresponding original image, we first compute the quadtree representation of the original image using a fixed scanning order at every internal node. We call this the *canonical quadtree representation* of the image. At each level, we try to find a leaf node having a unique gray level with respect to the other leaf nodes at the same level. The location of this leaf node in the encrypted quadtree also gives the locations of all ancestor nodes in the encrypted quadtree. With this information, we can restrict the scanning orders at the ancestors of this leaf node by fixing the order of the branch. If there is one such unique leaf node in three of the four quadrants, the scanning order at the root node is completely determined.

The above technique can be generalized as follows. Suppose we are given the canonical quadtree and encrypted quadtree representations of the image. For each subtree rooted at an internal node, we compute a signature that is permutation-invariant—a signature that remains the same regardless of the scanning order used at each internal node of the subtree. One example of such a signature is the number of each type of nodes at each level. The types of nodes considered are internal nodes and the different types of leaf nodes classified by their associated gray levels. An attack can be carried out by matching the four branches of the canonical quadtree to the corresponding branches in the encrypted quadtree at each node, starting at the root. The matching is done by comparing the signatures of the branches. If all four signatures are different, the scanning order at the node is completely determined. If some signatures are identical, we can only restrict the number of possible scan orders at the node. Unless all four signatures are identical, there will always be a reduction in the number of possibilities at each internal node.

If we are given multiple plaintext-ciphertext pairs, each pair can be used to determine a set of possible keys and the actual encryption key must belong to the intersection of the sets. By applying our known-plaintext attack on each plaintext-ciphertext pair, the search space is reduced. This may make exhaustive search feasible.

3.5.4 A Chosen-Plaintext Attack

The known-plaintext attack can be converted into an efficient chosen-plaintext attack. In this attack, we create input images such that the signature of each quadrant is different. By applying the known-plaintext attack to the encrypted output, we can completely determine the encryption key.

Suppose the quadtree image compression algorithm takes $2k$-bit images as input, and that we want to recover the key used to encrypt an image of dimensions $2^n \times 2^n$. If $n \leq k$, we construct an input image in which each pixel has a different gray level. The signature of each branch must be different, and the entire encryption key can be determined. If $n > k$, we construct an input image consisting of blocks of dimensions

Table 3.2: Summary of related algorithms.

Authors	Overhead	Attacks	Compression Performance
Jones [14]	negligible	chosen-plaintext	unaffected
Liu *et al.* [23]	twice as slow	unknown	2% worse
Matias and Shamir [25]	negligible	ciphertext-only known-plaintext chosen-plaintext	unaffected
Bourbakis and Alexopoulos [2]	negligible	chosen-plaintext	unknown
Chang and Liu [3]	negligible	ineffective keys known-plaintext chosen-plaintext	unaffected

$2^{n-k} \times 2^{n-k}$ arranged in an array of dimensions $2^k \times 2^k$. By assigning each block a distinct gray level, the scanning orders of the first k levels of the tree are completely determined. We can recursively apply the attack on each of the subtrees having a root at level $k-1$ by subdividing each block into at most 2^{2k} blocks, with unique gray levels within the parent block. Each encryption allows us to determine the scanning orders used for the internal nodes on k levels, so that $\lceil n/k \rceil$ encryptions are needed to completely determine the encryption key. For 8-bit images of dimensions 512×512, we have $n = 9$ and $k = 4$, so that only three encryptions are needed. Consequently, the chosen-plaintext attack is very efficient. In fact, only one encryption is needed to reconstruct an approximation of resolution $2^k \times 2^k$.

3.6 Summary

We summarize several proposals related to the work done in this thesis in Table 3.2. Jones' proposal on concealing the adaptive model in Huffman coding or arithmetic coding is vulnerable to chosen-plaintext attacks. The combined arithmetic coding and encryption algorithm by Liu *et al.* appears to be secure but is twice as slow as arithmetic coding alone. Image or video scrambling algorithms based on scan patterns, such as that proposed by Matias and Shamir, and that by Bourbakis and Alexopoulos, are vulnerable to different attacks. Finally, the quadtree image compression and encryption algorithm by Chang and Liu is vulnerable to many forms of attacks. We can conclude that none of the proposals satisfactorily solves the problem of reducing the overall compression and encryption processing time.

Chapter 4

Partial Encryption for Image and Video Communication

In this chapter, we describe the proposed approach to secure image and video communication. This is followed by a discussion on the advantages of this approach and evaluation criteria for partial encryption schemes. Finally, a list of potential applications of partial encryption is given. The idea of partial encryption was first proposed by the author in [4, 22].

4.1 Description

Many compression algorithms for multimedia data decompose their input into a number of different logical parts. Run-length encoding produces the run lengths and the parameters describing each run. Region-based image and video compression produces the shapes and locations of the regions as well as the parameters such as intensities or textures describing the regions. Transform coding algorithms such as those given by the Joint Photographic Experts Group (JPEG) [29] and the Moving Picture Experts Group (MPEG) [26] produce coefficients corresponding to the chosen basis functions that represent components of different frequencies.

These algorithms all have *important parts* that provide a significant amount of information about the original data. The remaining parts may not provide much information without the important parts. If this is the case, the remaining parts

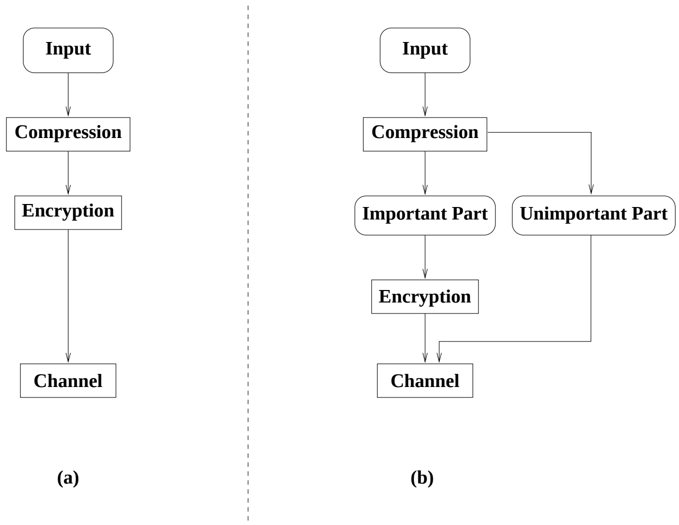


Figure 4.1: A comparison between (a) the traditional approach to secure image and video communication and (b) the proposed approach.

are called the *unimportant parts*. For simplicity, we consider all important parts as one important part. The remaining parts are grouped into one unimportant part, provided that it does not provide significant information about the original data.

In run-length encoding, the run lengths provide information on how the signal is partitioned and so it is important. On the other hand, the parameters describing the runs may not reveal much information about the original signal. We can only determine the runs that are present in the signal, but the lengths and the locations of the runs may still be unknown. In region-based compression, the shapes and locations of the regions are important while the parameters describing the regions may not be important. In transform coding, the low-frequency coefficients are important but the high-frequency coefficients may not be as important.

Since it is difficult to obtain information from the unimportant part alone, we propose a partial encryption approach in which only the important part is encrypted. A secure encryption algorithm is chosen to encrypt the important part. When the important part is small compared to the total output of the compression algorithm, a significant reduction in encryption and decryption time can be achieved. Figure 4.1 illustrates the difference between the proposed approach and the traditional approach,

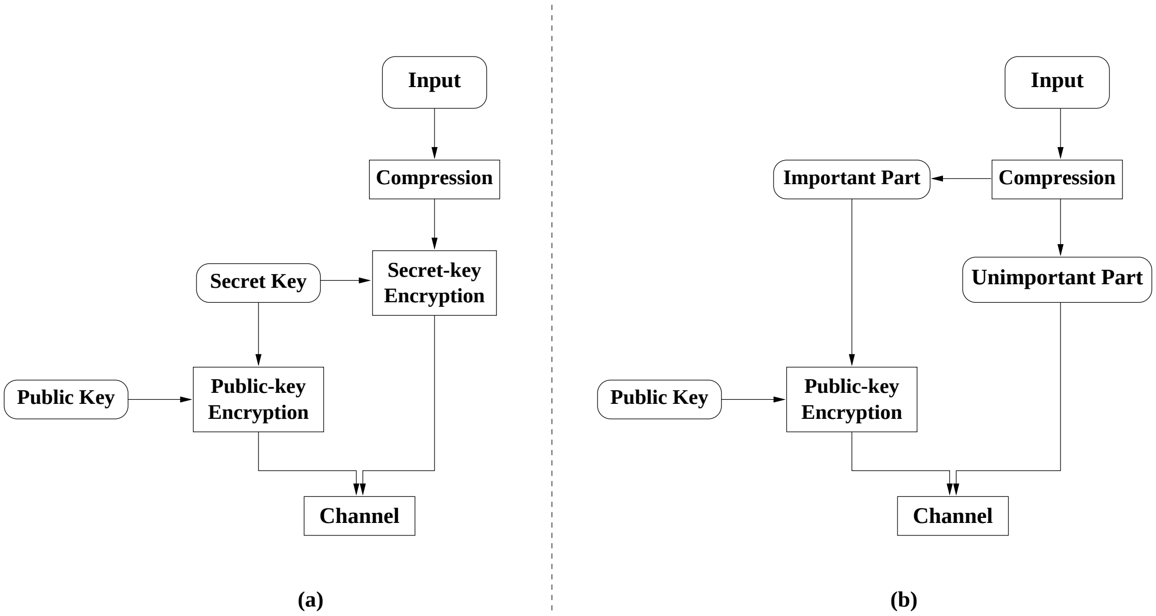


Figure 4.2: A comparison between (a) the traditional approach to secure image and video communication and (b) the proposed approach when public-key encryption can be applied directly to the important part.

in which the entire output of the compression algorithm is encrypted.

4.2 Advantages of Partial Encryption

The proposed approach has several advantages. First, it reduces the encryption and decryption time in the communication process. Without the reduction, the overhead for encryption and decryption often makes real-time secure video communication impossible. If the important part is encrypted while the unimportant part is transmitted in parallel, the encryption time may become negligible. Although the important part must be decrypted before decompression can take place, the total decryption time is reduced. Since the encoder is often much slower than the decoder in many multimedia compression algorithms, our approach can be used to help the encoder maintain the transmission rate that can be handled by the decoder.

The important part may be so small that public-key encryption algorithms such as the RSA algorithm [30] can be applied directly to the important part, as illustrated in Figure 4.2. The important part takes the role of the secret key in the traditional approach. Since secret-key encryption is not necessary, the implementation and operation costs for secret-key encryption are completely eliminated.

Partial encryption can also be viewed as a way to prioritize information, with the additional constraint that the unimportant parts alone do not reveal much information about the original data. We may allocate more bandwidth for the important part or pay more attention to the important part in error-detection or error-correction coding. This approach may also be used for progressive transmission, such that the important part is transmitted first and the unimportant part is transmitted only if necessary.

Partial encryption must be secure, such that it is computationally infeasible to recover the original data even when the unimportant part is known. Compression algorithms generally attempt to decompose their input into statistically uncorrelated parts to facilitate efficient encoding. If we separate the important part and the unimportant part based on these decompositions, the two parts will be almost uncorrelated. Although the two parts may still be statistically dependent, it may not be easy to take advantage of the dependence in an attack. As a result, if the encrypted important part and the unimportant part are known, ciphertext-only attacks on our scheme may not be significantly easier than ciphertext-only attacks on the chosen encryption algorithm. Known-plaintext and chosen-plaintext attacks on our scheme are as difficult as those on the chosen encryption algorithm because the knowledge of the unimportant part does not alter the probability of a key being the correct one. Part of this work has been published for two years [4, 22], and no attacks have been found.

Finally, the proposed approach is a technique that can be applied to many compression algorithms. As compression technology advances, we may apply this technique to the new algorithms proposed. Our approach is general, and it is not limited by the current technology in compression.

4.3 Evaluation of Partial Encryption Schemes

To evaluate each of our proposed partial encryption schemes, three aspects are examined:

Reduction in encryption and decryption time. Each scheme is evaluated by the amount of reduction in encryption and decryption time. Equivalently, each scheme can be evaluated by examining the size of the important part relative to the total output of the compression algorithm. The smaller it is, the more we save by encrypting only the important part.

Compression performance. Compression performance of the chosen compression algorithm must not be significantly affected by our scheme. Otherwise, the increase in transmission time may offset the reduction in encryption and decryption time. We will compare the compression performance of the original compression algorithm and that of the modified algorithm.

Security. Some justification on the security of each proposed scheme is provided. Unfortunately, it is often difficult, if not impossible, to prove that a scheme is secure. We will try to give some possible attacks and countermeasures, or show some properties that make our schemes difficult to cryptanalyze. This is by no means a proof of security. However, the only acceptable way to test the security of our schemes is to publish them and invite others to cryptanalyze it as others have done.

4.4 Potential Applications

Partial encryption and decomposition from an encryption perspective may have applications in addition to the reduction of encryption and decryption time in image and video communication. We list a few potential applications below.

Secure storage. Clearly, the partial encryption approach for secure communication can be applied to secure storage. Instead of transmitting encoded data in a communication channel, the encoded data is written to files. The files can later be decoded. The processing of encrypted images and video sequences requires the decryption of the stored data and possibly the encryption of the results. Partial encryption may reduce processing time significantly.

Other types of media. While this thesis focuses on image and video communication, partial encryption may also be applicable to other types of media such as text and audio.

Distributed image and video retrieval. The important part may be used to identify an image or a video sequence in a database, and the unimportant part can be transmitted on demand. The collection of the important parts of the available images forms a catalogue that is stored in each local computer. The catalogue is relatively small, so that it can be stored locally. When a particular image or video sequence is desired, the user requests a server to send the remaining parts unencrypted. Without the catalogue, eavesdroppers cannot decode the data. Only subscribed users have access to the catalogue, so unauthorized access is effectively denied. The implementation and operating costs of encryption and decryption are avoided. Since the remaining part is not required until a complete reconstruction is desired, data transmission can be totally eliminated when the user browses through the catalogue.

Secure broadcasting. In secure broadcasting, it may be necessary to encrypt the transmission to each user using a different key. For example, when the cost of the service is determined by the amount of time the service is used, separate encryption keys prevent users from subscribing for a short time simply to obtain the key for decrypting the transmission to another user. In this case, the broadcasting server has to encrypt a large volume of data, and this type of "personalized" encryption is infeasible without the reduction in encryption time offered by partial encryption.

Progressive transmission. In a progressive transmission scheme, a low-quality approximation is first transmitted and the quality of the approximation gradually improves. The user is given the opportunity to stop the transmission as soon as the quality is sufficient. If the important part can be used to construct a low-quality approximation of the original data, partial encryption can be used in progressive transmission by transmitting the important part first.

Chapter 5

Partial Encryption of Images

In this chapter, we examine the applicability of partial encryption to several image compression algorithms. We examine transform-based algorithms, quadtree algorithms, and wavelet compression algorithms based on zerotrees. Finally, we discuss the applicability of partial encryption to other image compression algorithms.

5.1 Transform-based Algorithms

In this section, we discuss one obvious decomposition of the transform coefficients in transform-based image compression algorithms. We use the image compression algorithm by the Joint Photographic Experts Group (JPEG) [29] as a specific example, but a similar argument can be applied to other transform-based compression algorithms [5].

The JPEG algorithm divides an image into 8×8 blocks, each of which is treated as a 64-dimensional vector. A change of basis transformation is applied so that the resulting vector contains the coefficients for expressing the input as a linear combination of the basis vectors. In JPEG, the discrete cosine transform (DCT) is used, and the basis functions correspond to two-dimensional functions of various spatial frequencies. The coefficient of the lowest frequency is called the DC coefficient, and the other coefficients are called the AC coefficients. Each coefficient is quantized independently before coding, and most of the high-frequency coefficients are small after

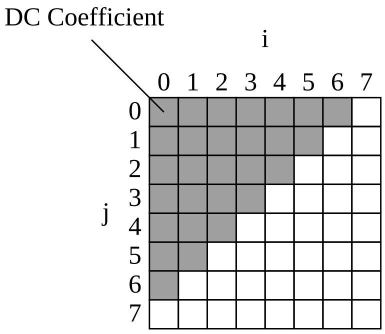


Figure 5.1: The encrypted DCT coefficients (shaded) with $N = 7$.

quantization. The discrete cosine transform has a decorrelation efficiency of 98.05% on each row and column of the block [5]. As a result, the coefficients are almost uncorrelated.

The low-frequency coefficients can be used to construct a blurred approximation to the original image, and so they must be encrypted. On the other hand, high-frequency coefficients only contribute to the fine details of the image. It may be possible to encrypt only the low-frequency coefficients. We need to determine the frequency cut-off between the important part and the remaining part.

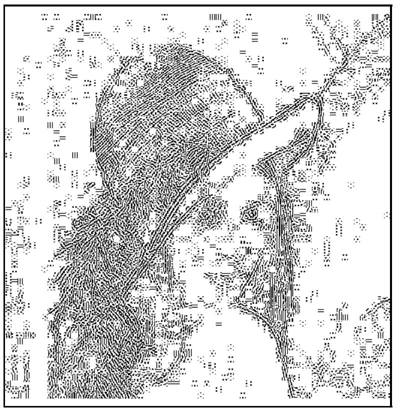
Let $\{f_i\}_{i=0}^{7}$ be an increasing sequence of frequencies, and $c_{i,j}$ be the coefficient, where $0 \le i, j < 8$, corresponding to the basis function with horizontal and vertical frequencies f_i and f_j, respectively. Since we must encrypt the low-frequency coefficients, we propose that the coefficient $c_{i,j}$ should be encrypted whenever $i + j < N$, where $1 \le N < 2 \times (8 - 1) = 14$. This is illustrated in Figure 5.1.

While the unencrypted high-frequency coefficients provide little information about the original 8×8 block, they indicate the presence of high-frequency components in the block. When the image blocks are considered together, the high-frequency coefficients alone often show the outline of objects in the image, as illustrated in Figure 5.2. Another problem is that most of the high-frequency coefficients are small, so that the number of bits required to represent the low-frequency coefficients is much higher than that required to represent the high-frequency coefficients. As a result,

(a) Original image

(b) $N = 3$

(c) $N = 4$



(d) $N = 5$

(e) $N = 6$

(f) $N = 7$

Figure 5.2: Reconstructed images (after contrast enhancement) using only the unencrypted high-frequency coefficients.

the encrypted important part is much larger than the unencrypted part.

Although the important part and the remaining part are almost uncorrelated, the decomposition of coefficients by frequency is not suitable for partial encryption. The important part is too large and it reveals too much information about the original image. An important point is that having almost uncorrelated important part and remaining part is not enough to ensure security in a partial encryption scheme.

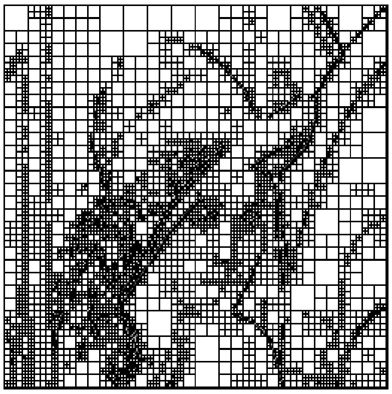
5.2 Quadtree Compression Algorithms

In quadtree image compression, two logical parts are produced—the quadtree and the parameters describing each block. We focus on the case in which the only parameter used to describe each block is the average intensity. Similar arguments apply to other types of parameters, and comments are made as necessary.

If the quadtree is known, we can reconstruct the quadtree decomposition to obtain outlines of objects in the original image, as illustrated in Figure 5.3. On the other hand, having only the intensity of each block does not allow us to learn much about the original image, as the location and size of each block are unknown. We propose a partial encryption scheme in which only the quadtree structure is encrypted, and the block intensities are left unencrypted. We refer to the block intensities as the *leaf values*, as each intensity corresponds to a leaf node in the quadtree. Clearly, the compression performance is unaffected because we are simply decomposing the output. The total size of the output remains the same.

The quadtree partial encryption scheme can be used for both lossless compression and lossy compression. In lossless quadtree compression, each leaf value is represented by the same number of bits. In lossy compression, however, the number of bits used to represent each leaf value is different. When a block is large, it is important to accurately represent its intensity. Thus, the leaf values corresponding to leaf nodes at a lower level are often represented by more bits than those corresponding to leaf nodes at a higher level. In this thesis, we concentrate on the bit allocation used by

(a) Original image



(b) Quadtree decomposition

Figure 5.3: Quadtree decomposition of an image.

Shusterman and Feder [36]:

$$b_i = \frac{1}{2} \log \left(\frac{\sigma_i^2 L_{qt}}{4^i D} \right), \tag{5.1}$$

where b_i is the number of bits used to represent each leaf value at level i, σ_i^2 is the variance of the leaf values at level i, L_{qt} is the total number of leaf nodes in the quadtree, and D is a constant specified by the user to control the bit rate. It is important that the decoder also has the b_i values in order to separate the individual leaf values, but the other parameters can be discarded once the bit allocation is computed. Since the bit allocation can be represented by very few bits, it is also included in the encrypted part.

5.2.1 Leaf Values Ordering

The leaf values must be transmitted in some order, and we now introduce two orderings of the leaf values. It will be shown that the ordering is crucial to the security of the partial encryption scheme. The quadtree in Figure 5.4 is used to illustrate the orderings in the description below. We assume that the four branches of each node in the quadtree correspond to the NW, SW, SE, and NE quadrants in this order. We also assume that a black leaf node has a leaf value of 1, and a white leaf node has a leaf value of 0.

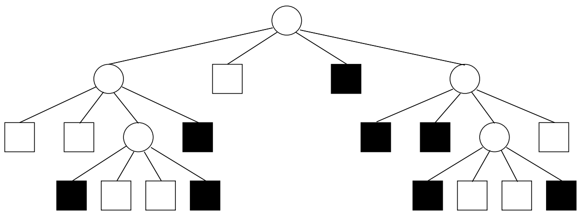


Figure 5.4: An example quadtree for a binary image.

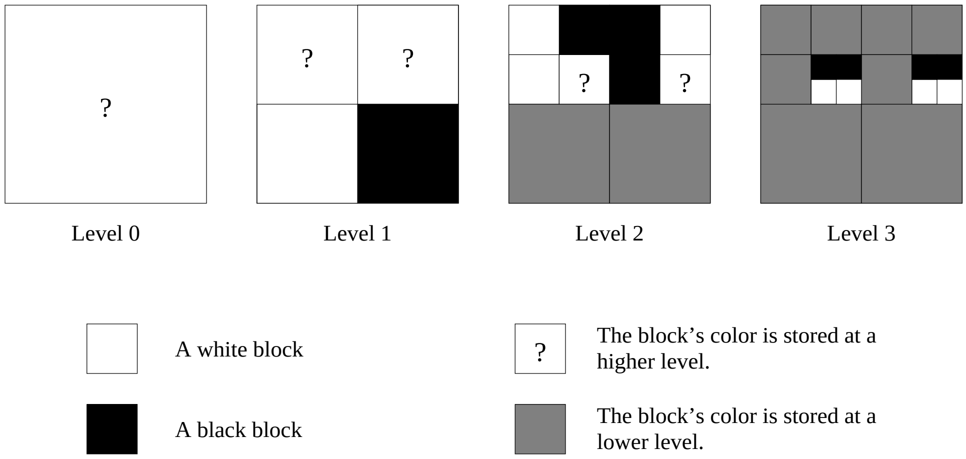


Figure 5.5: The spatial locations of the blocks at each level of the quadtree in Figure 5.4.

The first leaf values ordering, Leaf Ordering I, is induced by the inorder traversal of the quadtree [17]. In this ordering, the leaf values are encoded as 0010011011110010.

In Leaf Ordering II, the leaf values are encoded one level at a time. At each level, the blocks corresponding to the leaf values are ordered by the raster scan—each row of blocks are ordered left to right, and the rows are ordered from top to bottom. The leaf values are ordered according to the order of the corresponding blocks. This is illustrated in Figure 5.5, and the leaf values are encoded as:

Level 0: empty

Level 1: 01

Level 2: 011001

Level 3: 11110000

The leaf values of the levels are concatenated, and no special encoding is needed to separate each level if the quadtree is also known. We concatenate the leaf values from the highest level to the lowest level. Thus, the final encoded output is 1111000001100101.

5.2.2 Properties of Quadtrees

Several properties of quadtrees are needed in the analysis of the quadtree partial encryption scheme. Many of these results are generalized from the results and exercises for binary trees given in Section 2.3 of [17]. A proof of each statement is also given here.

Lemma 5.2.1 *A non-empty quadtree has* $4k + 1$ *nodes where* k *is a non-negative integer. In addition, it has* k *internal nodes, and* $3k + 1$ *leaf nodes.*

Proof. Any non-empty quadtree can be constructed from a single root node by repeatedly adding four children to an appropriate leaf node. Therefore, the quadtree has $4k + 1$ nodes where k is the number of times four children are added to any leaf node. Each addition converts one leaf node to an internal node. Thus, there are k internal nodes, and hence $4k + 1 - k = 3k + 1$ leaf nodes. □

Theorem 5.2.2 *There is a one-to-one correspondence between quadtrees having* $4k+1$ *nodes and a height of* $h + 1$ *and 4-ary trees having* k *nodes and a height of* h.

Proof. Let T be a 4-ary tree having k nodes and a height of h. To any node having fewer than four children, we add an appropriate number of leaf nodes as children to make the resulting tree, T', a quadtree. Each of the k nodes in T are internal nodes in T'. By Lemma 5.2.1, T' is a quadtree with $4k+1$ nodes. It is clear that the height of T' is $h + 1$, and that T' is unique.

Conversely, given a quadtree T having $4k+1$ nodes and a height of $h+1$, we remove all leaf nodes of T to obtain another tree T'. Clearly, T' is a unique 4-ary tree of height h. Only the internal nodes of T remain in T', and hence T' has k nodes. □

The following corollary is a straightforward consequence of the above theorem.

Corollary 5.2.3 *The number of quadtrees having $3k+1$ leaf nodes and a maximum height of $h+1$ is the same as the number of 4-ary trees having k nodes and a maximum height of h.*

Let $a_{k,h}$ denote the number of quadtrees having $3k+1$ leaf nodes and a maximum height of $h+1$. The following theorem provides a way to calculate $a_{k,h}$, and it is similar to the analysis of general trees found in [8].

Theorem 5.2.4 *Let $g_h(z) = \sum_k a_{k,h} z^k$ be the generating function of $a_{k,h}$ for a fixed h. Then,*

$$g_h(z) = \begin{cases} 1+z & \text{if } h = 0, \\ 1+z(g_{h-1}(z))^4 & \text{if } h > 0. \end{cases} \tag{5.2}$$

Proof. By Corollary 5.2.3, $a_{k,h}$ is also the number of 4-ary trees having k nodes and a maximum height of h. It suffices to prove the theorem by considering 4-ary trees.

For $h = 0$, it is clear that $a_{0,h} = a_{1,h} = 1$ and $a_{k,h} = 0$ for $k > 1$. Now, suppose that $h > 0$. A non-empty 4-ary tree can be formed by adding to a root node four independent branches of maximum height $h-1$. Thus, the factor z in the second term corresponds to the root node, and the factor $(g_{h-1}(z))^4$ corresponds to the four branches. The first term is added for the unique empty 4-ary tree. It follows that the theorem holds. □

5.2.3 File Sizes

The size of the important part must be small relative to the total output for the encryption and decryption time to be significantly reduced. We first give an analysis of the size of the quadtree relative to the leaf values for both lossless and lossy quadtree compression. It is followed by experimental results on the quadtree compression algorithm by Knipe [15].

Each node of the quadtree is either an internal node or a leaf node. Thus, a quadtree may be represented by one bit per node. In fact, nodes corresponding to 1×1 blocks need not be represented at all because they must be leaf nodes. To simplify the calculations, we will assume that the quadtree is represented by one bit for each node, even for the leaf nodes corresponding to 1×1 blocks. In this way, we obtain upper bounds on the quadtree size.

We have seen that a quadtree has $4k+1$ nodes for some $k \geq 0$. The size of the quadtree is $4k+1$ bits, $3k+1$ of which correspond to leaf nodes. In the case of lossless compression on a b-bit image, the total size of the leaf values is $b(3k+1)$ bits. Upper bounds on the relative size of the quadtree are given in Table 5.1 for various values of b. For example, when $b = 8$, the size of the quadtree relative to the total output is at most

$$\begin{aligned} & \frac{4k+1}{8(3k+1)+4k+1} \\ =& \frac{4k+1}{7(4k+1)+2} \\ \approx& \frac{1}{7} \\ =& 14.3\%. \end{aligned} \tag{5.3}$$

For lossy compression, the situation is more complicated because the number of bits used for each leaf value is different. The bit allocation for leaf values at a lower level is higher than that for leaf values at a higher level. Thus, the relative quadtree size is smaller if there are many leaf nodes at low levels. Hence, an image that is easily compressed should have a relatively small quadtree. It is difficult to analyze the size of the quadtree without the knowledge of the bit allocation as well as the distribution of leaf nodes among the different levels. Instead, results were collected from experiments performed on test images, using an implementation of the quadtree compression algorithm by Knipe [15].

The results are shown in Table 5.2. The results show that the relative quadtree size

Table 5.1: Approximate upper bounds on the relative quadtree size in lossless quadtree compression on b-bit images.

b	Quadtree Size (%)
1	57.1%
2	40.0%
3	30.8%
4	25.0%
5	21.1%
6	18.2%
7	16.0%
8	14.3%
9	12.9%
10	11.8%

of all but one image is between 13%–27%. The only exception is the "washsat" image, which has a relative quadtree size of 56.1% (highlighted in Table 5.2). The reason is that "washsat" is a low-contrast image, so that the values σ_i^2 in Equation (5.1) are small. Hence, many leaf values are represented by only one bit. This is also the reason for the low bit rate achieved by the compression algorithm. Note that if each leaf value is represented by at least one bit, then the relative quadtree size is at most 57.1% as shown in Table 5.1.

The analysis is similar when other parameters are used to describe the blocks associated with the leaf nodes. For example, Strobach [38] used the three parameters a, b, c to describe each block by the first-order model $f(x, y) = a + bx + cy$. The number of bits used to represent each set of parameters can be used to predict the quadtree size. It is shown in [38] that at approximately 0.5 bpp, the quadtree size is 11.7% for the "boat" image, and 8.4% for the "lena" image.

5.2.4 Security

The security of the quadtree partial encryption scheme is now examined. We first discuss the infeasibility of attacks based on exhaustive tree enumeration. Next, we present attacks on lossless and lossy quadtree compression using Leaf Ordering I.

Table 5.2: Relative quadtree size in lossy quadtree compression.

Image	Dimensions	bpp	Quadtree Size (bytes)	Quadtree Size (%)
airfield	512×512	1.323	6244	14.4%
airplane	512×512	0.530	3601	20.7%
barbara	512×512	1.210	5524	13.9%
bay	256×256	0.933	1371	17.9%
bird	256×256	0.370	643	21.2%
boat	512×512	0.673	4352	19.7%
bridge	256×256	1.826	2178	14.6%
camera	256×256	0.877	1035	14.4%
couple	512×512	0.956	5064	16.2%
crowd	512×512	0.813	5232	19.6%
festung	512×512	0.655	3922	18.3%
goldhill	512×512	0.631	4825	23.3%
io	512×512	0.784	4188	16.3%
lax	512×512	1.155	6331	16.7%
lena	512×512	0.547	3537	19.7%
man	512×512	0.852	4974	17.8%
mandrill	512×512	1.913	8168	13.0%
peppers	512×512	0.516	3388	20.0%
sailboat	512×512	0.671	4621	21.0%
shepherd	512×512	1.502	7083	14.4%
sunset	256×256	0.542	859	19.4%
washsat	512×512	**0.132**	**2420**	**56.1%**
woman1	512×512	0.757	4186	16.9%
woman2	512×512	0.296	2290	23.6%
zelda	512×512	0.344	3027	26.8%

Finally, we discuss the difficulties in cryptanalyzing partial encryption based on Leaf Ordering II.

Attacks by Tree Enumeration

Suppose we know that the dimensions of the image is $2^n \times 2^n$, then the maximum height of the quadtree is n. In the case of lossless compression, the number of bits used for each leaf value is fixed. Hence, the number of leaf nodes in the quadtree can be obtained. In the case of lossy compression, the number of leaf nodes may not be known exactly, but we may be successful in determining a range of possible values.

Given the number of leaf nodes, L, and the maximum height, n, of the quadtree, a cryptanalyst may devise an algorithm that generates all possible quadtrees with the specified parameters, and match each tree with the leaf values until the correct image is found. The number of such quadtrees is $a_{k,h}$ where $k = (L-1)/3$ and $h = n-1$. The values $\log_2 a_{k,h}$ can be computed by the recurrence relation in Theorem 5.2.4, and are plotted for $h = 4, 5, 6$ in Figure 5.6. The plots show that unless k is very small or very large, the number of possible quadtrees is very large. As a result, exhaustive tree enumeration is infeasible unless the quadtree is almost empty or almost complete. In Table 5.3, we show that this does not occur for typical images. The results are obtained using lossy quadtree compression, and $k_{\max}$ is the value of k for a complete quadtree.

Suppose now that we also have the knowledge of the number of leaf nodes at each level. Then, we may reduce the number of possible quadtrees by examining the locations of the leaf nodes at each level. Let l_i be the number of leaf nodes at level i, and m_i be the number of possible leaf locations at level i. Then

$$\begin{aligned} m_i &= \frac{2^{2n} - \sum_{j=0}^{i-1} l_j 2^{2(n-j)}}{2^{2(n-i)}} \\ &= 2^{2i} - \sum_{j=0}^{i-1} l_j 2^{2(i-j)}. \end{aligned} \tag{5.4}$$

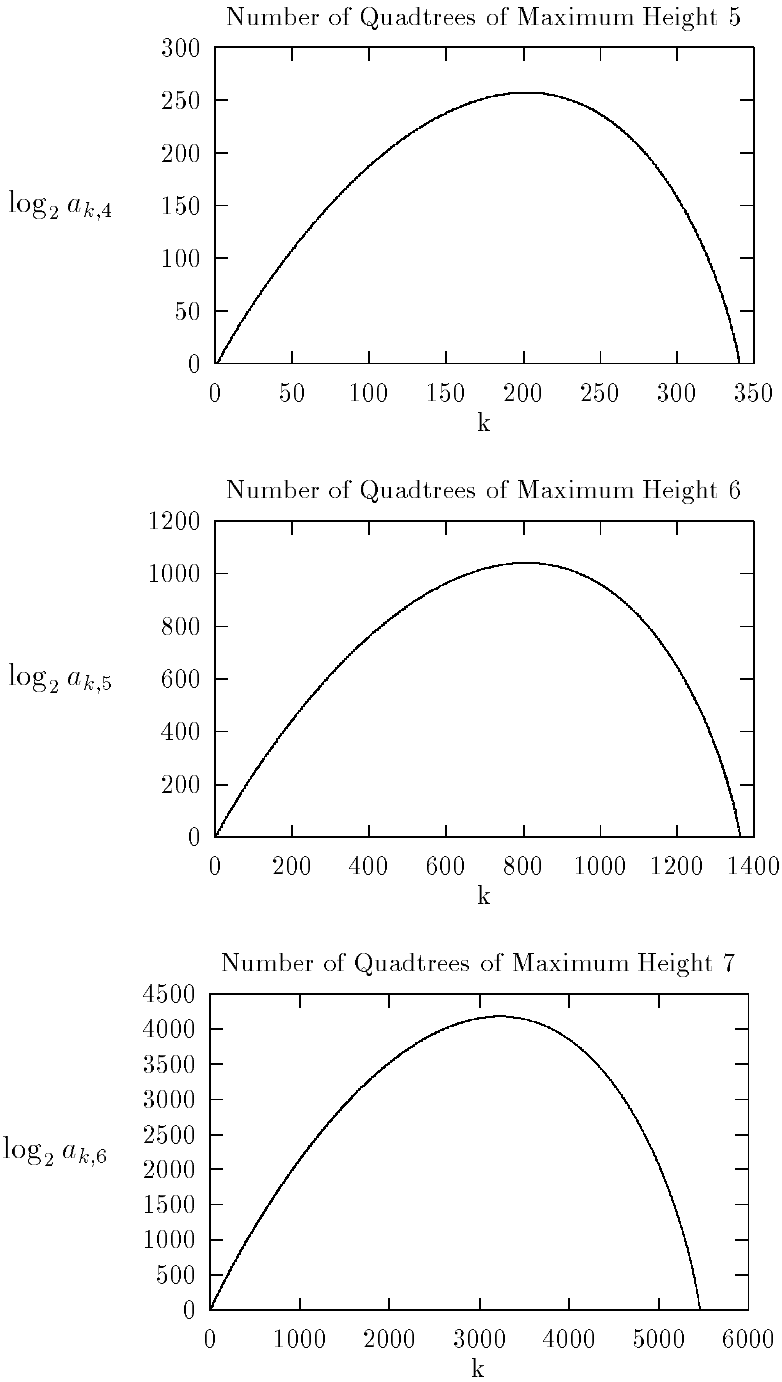


Figure 5.6: Plots of $\log_2 a_{k,h}$ for various values of h.

Table 5.3: Values of h and k for various images.

Image	Dimensions	h	k	$k_{\max}$
airfield	512×512	8	23551	87381
airplane	512×512	8	12310	87381
barbara	512×512	8	24839	87381
bay	256×256	7	5246	21845
bird	256×256	7	1798	21845
boat	512×512	8	14533	87381
bridge	256×256	7	8680	21845
camera	256×256	7	4129	21845
couple	512×512	8	16811	87381
crowd	512×512	8	16237	87381
festung	512×512	8	12931	87381
goldhill	512×512	8	13697	87381
io	512×512	8	13864	87381
lax	512×512	8	26455	87381
lena	512×512	8	10274	87381
man	512×512	8	15174	87381
mandrill	512×512	8	39837	87381
peppers	512×512	8	9891	87381
sailboat	512×512	8	15724	87381
shepherd	512×512	8	25507	87381
sunset	256×256	7	2230	21845
washsat	512×512	8	5524	87381
woman1	512×512	8	13942	87381
woman2	512×512	8	4902	87381
zelda	512×512	8	6752	87381

Table 5.4: Number of quadtrees for the "lena" image given the number of leaf nodes at each level.

i	l_i	m_i	$\log_2 \binom{m_i}{l_i}$
0	0	1	0
1	0	4	0
2	0	16	0
3	0	64	0
4	1	256	8
5	115	1020	514
6	1511	3620	3542
7	4747	8436	8333
8	11525	14756	11182
9	12924	12924	0
Total			23579

The number of possible quadtrees is then

$$\prod_{i=0}^{n} \binom{m_i}{l_i}. \tag{5.5}$$

Table 5.4 shows the number of possible quadtrees for the "lena" image. Even with the extra information on l_i, the number of possible quadtrees is approximately 2^{23579}. Other images give similar results. We conclude that it is computationally infeasible to attack our partial encryption scheme using exhaustive tree enumeration.

Attacks on Lossless Compression Using Leaf Ordering I

The partial encryption scheme based on lossless quadtree compression using Leaf Ordering I has two properties that facilitate cryptanalysis. First, the leaf values for sibling nodes are close together in the encoded leaf values. Moreover, the number of bits used for each leaf value is fixed, so that it is easy to separate the encoded data into individual leaf values.

Let us assume that the four branches of a node correspond to the NW, SW, SE, and NE quadrants. It follows that the first leaf value corresponds to a block containing the NW corner of the image, and the last leaf value corresponds to a block containing the NE corner of the image. Thus, two pixels are revealed.

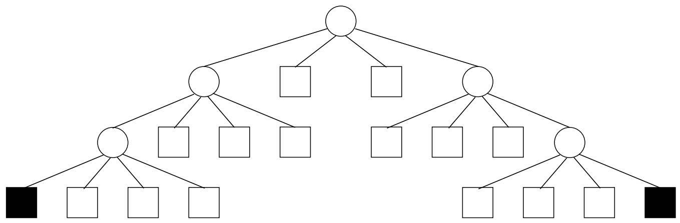


Figure 5.7: A quadtree having a run of length 14.

A *run*, or a sequence of identical leaf values, in the encoded data can be used in a cryptanalysis. An important property of the encoded data is that four leaf values in a run cannot correspond to sibling nodes; otherwise, they would have been merged into one single node. As a result, long runs reduce the number of quadtrees that need to be examined in an exhaustive search. In Figure 5.7, a quadtree having a run of length 14 is shown. If we examine the levels of the leaf nodes in this run, it does not change direction (decreasing or increasing) many times. This follows from the fact that we must encounter four sibling leaf nodes in order for an increase to be changed into a decrease, and these four sibling leaf nodes cannot occur within the run.

Long runs may be used to recover homogeneous regions if the run starts at a NW leaf node. Suppose we have a run of length k starting at a NW leaf node of some subtree, and let r be the leaf value appearing in this run. A homogeneous region may be recovered as follows:

1. Set $i = 1$, and set (x, y) to the coordinates of the NW pixel of the block represented by this subtree.

2. While $i \leq k$ do

   (a) Write r into the image location at (x, y).

   (b) If (x, y) is a NE corner of some quadrant, then the values in the quadrant should be merged. If the dimensions of the merged quadrant is $2^m \times 2^m$, then set $i = i - 3m$.

   (c) Set (x, y) to the next image location using the ordering NW, SW, SE, NE. That is, the next pixel location visited by an inorder traversal of the complete quadtree.

(d) Set $i = i + 1$.

A similar procedure can be given for runs that end at a NE leaf node. While it may not be easy to classify the leaf nodes corresponding to the start and the end of a run in general, we may apply the above procedure to the runs at the beginning and the end of the encoded leaf values.

This procedure can also be used in a known-plaintext attack. If a known non-homogeneous image portion is located within some quadrant, quadtree compression can be performed on this portion to obtain the leaf values. The leaf values of the image portion would then be a substring of the leaf values of the entire image. The leaf value immediately following the substring is a NW leaf node, and the one immediately before the substring is a NE leaf node. Consequently, the above procedure can be used on long runs before and after the substring to extend the known image portion.

Although the start and the end of a run may not be classified, we may still obtain information about the leaf values in a run. Assume that we are given a run of length k. Then, we can determine the minimum dimensions of the blocks represented by the leaf values in the run as follows:

1. Set $i = k$, $j = 0$.
2. While $i > 0$ do
   (a) There are $\min(i, 6)$ leaf values in the run corresponding to blocks of minimum dimensions $2^j \times 2^j$.
   (b) Set $i = i - 6$, $j = j + 1$.

The only situation in which we obtain the true dimensions of the blocks occurs when the quadtree is skewed both to the left and to the right, but is empty in the middle.

An example is given in Figure 5.7. The statistics obtained from this procedure may be used to estimate the image size, the compression ratio, and the image histogram.

Although the above attacks cannot be used to obtain the entire image, we do not recommend using Leaf Ordering I for lossless compression. In particular, many computer generated images have a uniform background that includes the corners of the image. Long runs may be used to obtain homogeneous regions at the corners and may reveal the shapes of foreground objects.

Attacks on Lossy Compression Using Leaf Ordering I

When Leaf Ordering I is used with lossy compression, it is more difficult to use attacks based on long runs. First, the leaf values are represented using variable number of bits, so that it is difficult to obtain the individual leaf values. Moreover, it is more difficult to recognize a run as it may not contain identical leaf values.

In the case of lossy compression, however, it is crucial that the leaf values cannot be separated without the quadtree. Suppose that we have obtained the individual leaf values, and that the non-zero bit allocation for each level is distinct. After adding enough zeros, we obtain the values b_i ($i = 0, \ldots, n$) in some order. Since $n \leq 12$ for most images, we need to examine at most $12! < 2^{29}$ orderings of the b_i values to obtain the actual bit allocation. This is certainly feasible with today's technology. Once the correct order is obtained, the level at which each leaf value is located is also known. This information can be used to completely determine the quadtree. The level of the first leaf value gives complete information on the leftmost branch of the quadtree. The level of the second leaf value gives the information on the branch that follows immediately after the first one in inorder traversal. This can be applied successively to each leaf value until the quadtree is completely determined. Note that this attack is only possible if the non-zero bit allocation for each level is distinct.

As a result, we should ensure that no separators are inserted between the leaf values, and that the leaf values are not transmitted individually. Together with the undesirable property of Leaf Ordering I that the leaf values of sibling nodes are close together, we conclude that Leaf Ordering I should not be used in our partial encryption scheme for lossy compression.

Leaf Ordering II

Leaf Ordering II is more secure when used in either lossless compression and lossy compression. In the case of lossless compression, leaf values of sibling nodes may be far apart. In fact, the number of leaf values between the leaf values of two sibling nodes may be as many as the number of possible blocks in a row. Thus, it is more difficult to use the property that four sibling nodes must have different leaf values. The first and last leaf values do not correspond to fixed pixels in the image. Furthermore, the compressed output of a known image portion is generally not a substring of the leaf values of the entire image, so that the known portion cannot be extended.

In the case of lossy compression, Leaf Ordering II allows the individual leaf values to be separated without compromising security. The same attack based on bit allocation only gives the number of leaf nodes at each level. We have already seen that the number of possible quadtrees is very large even with this extra information, making exhaustive search computationally prohibitive.

Other Comments

Given a quadtree, there are many possible sets of leaf values that match with the quadtree. In particular, we can assign many different sets of intensities to each block provided that four sibling blocks do not have identical or similar gray levels. Conversely, given a set of leaf values, there are many possible quadtrees that match with the leaf values. If the distribution of the block sizes are the same but the block locations are changed, we can generate quadtrees that match with the same leaf values. In fact, the security of our partial encryption scheme depends on the last property. This many-to-many relationship between quadtrees and leaf values may be crucial to the security of our partial encryption scheme.

The raster scan used to define Leaf Ordering II may also be substituted by any other permutation of the blocks at each level, provided that it is difficult to determine whether any two given leaf values correspond to sibling nodes. We may wish to use a secret permutation to add more complexity to the cryptanalysis of the partial encryption scheme. However, the security of our partial encryption scheme still depends mainly on the hidden quadtree.

5.2.5 Implementation Issues and Complexity

Since Leaf Ordering I is not secure, only Leaf Ordering II is discussed below. Leaf Ordering II can easily be used in a bottom-up implementation of the quadtree image compression algorithm. At each level, the leaf nodes are examined for merging in the spatial order described by Leaf Ordering II. If a leaf node is not merged with its siblings, then the leaf value is transmitted. Since the algorithm must examine the leaf nodes at each level in some order, there is little computational difference among the different orders when a look-up table is used to specify the order. Thus, the overhead of using Leaf Ordering II is negligible.

5.3 Wavelet Compression Algorithms Based on Zerotrees

Wavelet compression algorithms based on zerotrees generally transmit the structure of the zerotrees in addition to the significant coefficients. For example, the Set Partitioning in Hierarchical Trees (SPIHT) compression algorithm [32] introduced in Chapter 2 transmits the significance of the coefficient sets which are trees of coefficients. This is similar to quadtree compression as it indicates whether a set needs to be decomposed further. Instead of homogeneity, insignificance is the factor for deciding whether a set is partitioned. In this section, we propose a partial encryption scheme for compression algorithms based on zerotrees. We will use the SPIHT compression algorithm in the discussion, but other algorithms based on zerotrees can also be used [7, 15, 16, 35].

The SPIHT algorithm uses the significance information of sets to determine the tree structures, and the execution of the algorithm depends strongly on the structure of the zerotrees. Even if a small amount of significance information at the beginning of the encoded data is incorrect, the algorithm cannot decode the image correctly [6, 32]. The compression algorithm produces many different types of bits—sign bits, refinement bits, significance of pixels, and significance of sets. The decompression algorithm must interpret each bit under the correct context. Incorrect significance bits may cause future bits to be misinterpreted, while incorrect sign bits or refinement bits do not.

The SPIHT algorithm produces two parts in the output—the significance information and other information related to coefficient magnitudes. We propose a partial encryption scheme that encrypts only the significance information related to pixels or sets in the two highest pyramid levels, as well as the parameter n that determines the initial threshold. Formally, we encrypt the significance information $S_n(i, j)$, $S_n(\mathcal{D}(i, j))$, and $S_n(\mathcal{L}(i, j))$ if and only if (i, j) is in the two highest pyramid levels. If the root level has dimensions 8×8, then the significance information is encrypted if and only if $0 \leq i, j < 16$. Figure 5.8 shows the encrypted pyramid levels for an 128×128 image. Clearly, the compression performance of the algorithm is unaffected.

We do not encrypt all significance information because the pixels and sets in other levels are produced by decomposing sets in the two highest pyramid levels. The states of the three lists LIP, LIS, and LSP are constantly updated by the algorithm, and the significance information is used to determine the way in which the lists are updated. If the states of the lists are incorrect at the beginning of the algorithm, it is difficult for the algorithm to recover from the error. Our goal is to encrypt enough of the significance information so that it is difficult for the cryptanalyst to determine the meaning of each unencrypted bit.

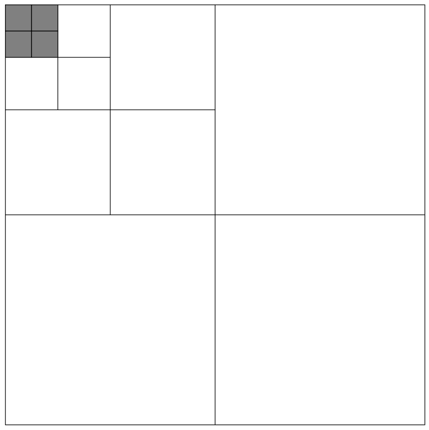


Figure 5.8: The encrypted pyramid levels for an 128×128 image.

This partial encryption scheme is similar in one aspect to the combined compression and encryption algorithms by Jones [14] and Liu *et al.* [23] described in Chapter 3. All three algorithms conceal their initial states. If the initial states are incorrect, it is difficult to correctly decode the data. Our partial encryption scheme differs from the other two in that the initial states cannot be chosen freely. In our case, the initial states are completely determined by the input image. As a result, the initial states must be encrypted. This partial encryption scheme is also similar to the quadtree partial encryption scheme proposed earlier. Instead of encrypting the entire tree structure, only the lowest levels of the trees are encrypted. The remaining information on the tree structure is mixed with the leaf values in the unencrypted part.

5.3.1 File Sizes

We now give some analytical results on the size of the encrypted important part. This is followed by some experimental results on our test images.

In the SPIHT algorithm, the size of the encoded data is directly controlled by the bit rate. The encoder stops the transmission as soon as the desired bit rate is reached. The bit rate also indirectly affects the number of sorting passes performed on the wavelet coefficients. If a pixel or a set in the two highest pyramid levels is considered for m sorting passes, then m bits are transmitted for its significance. From this, we may establish an upper bound and a lower bound on the size of the important part.

Theorem 5.3.1 *Let m be the number of sorting passes performed by the encoder, including the last one that may not be completed. Then, the size of the important part is at most $496m + 240$ bits.*

Proof. In the worst case where each coefficient in the two highest pyramid levels is 0, $S_n(i, j)$ is transmitted for each $c_{i,j}$ in the encrypted pyramid levels during each sorting pass. There are 256 coefficients in the two levels, so the significance of these coefficients gives at most 256 bits during each sorting pass. It follows that an upper bound on the number of bits contributed by the significance of the coefficients is $256m$.

To complete the proof, we observe that for any coordinates (i, j), the LIS cannot contain entries for both $\mathcal{D}(i, j)$ and $\mathcal{L}(i, j)$ at the same time. Once the $\mathcal{L}(i, j)$ entry is added to the LIS, the $\mathcal{D}(i, j)$ entry is removed. During Step 3(b) of each sorting pass, the encoder processes at most one of $\mathcal{D}(i, j)$ and $\mathcal{L}(i, j)$ except for the pass when the set $\mathcal{D}(i, j)$ is identified as significant, in which case both entries are processed. At most one bit is transmitted for each entry processed. Consequently, each (i, j) in the two highest pyramid levels may contribute up to $m + 1$ bits for $S_n(\mathcal{D}(i, j))$ and $S_n(\mathcal{L}(i, j))$. There are 240 coefficients in the two highest pyramid levels with descendants, and hence the number of bits contributed by $S_n(\mathcal{D}(i, j))$ and $S_n(\mathcal{L}(i, j))$ is at most $240(m + 1)$.

Thus, an upper bound for the size of the important part is $256m + 240(m + 1)$ bits, or $496m + 240$ bits. □

Theorem 5.3.2 *If at least two sorting passes are performed by the encoder, then the size of the important part is at least 160 bits.*

Proof. We first determine the contribution to the important part by $S_n(i,j)$, where (i,j) is in the highest two levels of the pyramid. During the first sorting pass, 64 bits are always encrypted regardless of the significance of the root level coefficients. If all the root level coefficients are significant with respect to the highest threshold, there is no output corresponding to the significance of the root level coefficients in the second sorting pass. Consequently, a lower bound on this part is 64 bits.

We now consider the contribution from $S_n(\mathcal{D}(i,j))$ and $S_n(\mathcal{L}(i,j))$ where $(i,j) \in \mathcal{H}$. At least one bit is transmitted for $S_n(\mathcal{D}(i,j))$ in Step 3(b)i of the first sorting pass, where $c_{i,j}$ is a coefficient with descendants in the root level. If $\mathcal{D}(i,j)$ is significant, then $\mathcal{L}(i,j)$ would be processed and an additional bit is contributed to the important part. If $\mathcal{D}(i,j)$ is not significant, another bit is transmitted for $S_n(\mathcal{D}(i,j))$ during the second sorting pass. In either case, at least 2 bits are transmitted during the first 2 sorting passes. Since there are 48 root level coefficients with descendants, at least 96 bits are contributed to the important part.

Therefore, the size of the important part is at least $64 + 96 = 160$ bits. $\square$

In most cases, 6 to 10 sorting passes are performed when the images are compressed at 0.80 bpp, giving an upper bound of 5200 bits on the important part. In practice, the actual size of the important part is usually much smaller. Table 5.5 shows the results obtained at various bit rates. The size of the important part is less than 2% of the total output for each image of dimensions 512×512, and less than 7% for each image of dimensions 256×256.

5.3.2 Security

Without the significance information of the two highest pyramid levels, the initial changes to the three lists LIS, LIP, and LSP are not known. By Theorem 5.3.2, at least 160 bits of significance information must be known to correctly decode the image. Thus, an exhaustive search would require testing at least 2^{160} possibilities. The embedded nature of the algorithm may allow fewer bits to be used to construct a low-

Table 5.5: The size of important part in the SPIHT algorithm.

Image	Dimensions	0.80 bpp		0.60 bpp		0.40 bpp	
		Bits	%	Bits	%	Bits	%
airfield	512×512	1653	0.8%	1653	1.1%	1642	1.6%
airplane	512×512	1710	0.8%	1704	1.1%	1680	1.6%
barbara	512×512	1722	0.8%	1716	1.1%	1709	1.6%
bay	256×256	1594	3.0%	1556	4.0%	1471	5.6%
bird	256×256	1895	3.6%	1830	4.7%	1717	6.6%
boat	512×512	1841	0.9%	1835	1.2%	1805	1.7%
bridge	256×256	1509	2.9%	1487	3.8%	1478	5.7%
camera	256×256	1752	3.3%	1724	4.4%	1656	6.3%
couple	512×512	1634	0.8%	1624	1.1%	1617	1.5%
crowd	512×512	1674	0.8%	1670	1.1%	1670	1.6%
festung	512×512	1770	0.8%	1744	1.1%	1720	1.6%
goldhill	512×512	1741	0.8%	1729	1.1%	1726	1.6%
io	512×512	1730	0.8%	1724	1.1%	1720	1.6%
lax	512×512	1561	0.7%	1561	1.0%	1535	1.4%
lena	512×512	1684	0.8%	1676	1.1%	1670	1.6%
man	512×512	1570	0.7%	1566	1.0%	1566	1.5%
mandrill	512×512	1533	0.7%	1516	1.0%	1510	1.4%
peppers	512×512	1647	0.8%	1647	1.0%	1647	1.6%
sailboat	512×512	1748	0.8%	1741	1.1%	1734	1.7%
shepherd	512×512	1596	0.8%	1596	1.0%	1590	1.5%
sunset	256×256	1769	3.4%	1761	4.5%	1723	6.6%
washsat	512×512	1435	0.7%	1435	0.9%	1435	1.4%
woman1	512×512	1658	0.8%	1657	1.1%	1651	1.6%
woman2	512×512	1884	0.9%	1884	1.2%	1883	1.8%
zelda	512×512	1808	0.9%	1808	1.2%	1805	1.7%

quality approximation, but experiments show that at least 160 bits of the important part must be known to give a very coarse approximation. This is illustrated in Figure 5.9, in which the decoder stops as soon as k encrypted bits are received.

Our experiments have shown that if the number of 1 bits in each sorting pass is correctly determined, the algorithm may still interpret the meaning of each bit correctly. While the locations of significant subtrees may be moved, the number of each type of entries in the three lists remains the same. This results in some distortion in the reconstructed image, but the image is still recognizable. However, it is difficult to take advantage of this property in an attack because the number of 1 bits and the total number of bits produced in each sorting pass are unknown.

It is difficult to cryptanalyze this partial encryption scheme for several reasons:

1. The unimportant part contains different types of bits that are mixed together. Without the significance information, one type of bits cannot be distinguished from another.

2. The bit at which a sorting pass or a refinement pass starts in the unimportant part is difficult to locate. It is difficult to identify the step at which each bit is produced in the algorithm.

3. The order in which the coefficients are examined in the refinement pass depends on the state of the LSP. If the initial updates to the LSP are not known, it is difficult to determine the order in which the coefficients are refined.

4. The encrypted significance information may span multiple sorting passes, making it difficult to infer the number of 1 bits in the first few sorting passes from the size of the important part.

5. There is a many-to-many relationship between the important part and the unimportant part. Given an important part, there are many unimportant parts that match with it. Namely, we may change the signs and all but the most significant

(a) Original image

(b) $k = 100$

(c) $k = 120$

(d) $k = 140$

(e) $k = 160$

(f) $k = 180$

Figure 5.9: Reconstructed images using only the first k encrypted bits.

bit of the coefficients and obtain the same important part. On the other hand, we may generate multiple important parts that match a given unimportant part by translating the significant branches.

6. The important parts of two similar images may be dramatically different. Substituting the important part of one image into another does not give a meaningful approximation of the original image.

5.3.3 Implementation Issues and Complexity

It is straightforward to add our partial encryption scheme to the SPIHT algorithm. We simply perform an additional comparison before each significance bit is transmitted. The comparison is performed on the coordinates (i, j) to determine whether the significance information is encrypted. In our experiments, we have not been able to detect any increase in running time due to the additional comparisons.

5.4 Summary of Proposed Schemes

We summarize the properties of the proposed partial encryption schemes in Table 5.6. In this table, QT-PE is the proposed quadtree partial encryption scheme, and SPIHT-PE is the proposed SPIHT partial encryption scheme. The "Amount Encrypted" column refers to the total amount of data that is encrypted by an independent encryption algorithm for typical images. The "Overhead" column indicates the impact on running time due to the additional processing for separating the important part from the unimportant part. The "Compression Performance" column indicates the impact on compression performance when partial encryption is used. In both cases, there is little effect in the compression performance and complexity of the compression algorithm. In addition, a relatively small amount of the total output needs to be encrypted. No attacks are known at this time, and the author feels that the partial encryption schemes are secure.

Table 5.6: Summary of proposed partial encryption schemes.

Scheme	Lossy or Lossless	Amount Encrypted	Overhead	Compression Performance
QT-PE	both	13-27%	negligible	unaffected
SPIHT-PE	lossy	2% (512×512) 7% (256×256)	negligible	unaffected

5.5 Other Image Compression Algorithms

We have examined the application of partial encryption to three different image compression algorithms. We conclude this chapter with a discussion on the applicability of partial encryption to other image compression algorithms.

Many region-based algorithms use tree structures to represent the regions of the image, and leaf values to represent the properties of the regions. For example, adaptive tree-structured segmentation [42] and B-tree triangular coding [9] both use binary trees to represent the regions. If we apply partial encryption to these algorithms and encrypt only the tree, we may apply similar analysis to obtain the file sizes and security of the resulting schemes. The low branching factor of binary trees affects the file sizes and the security of the partial encryption schemes. We have not examined these region-based compression algorithms in detail in this thesis.

There are also compression algorithms that use wavelet packets [31]. These algorithms differ from wavelet compression algorithms in that wavelet decomposition is not always performed recursively on the LL coefficient band. Any coefficient band may be chosen for decomposition. Since this decomposition can be represented by a variable tree structure, we may ask whether it is possible to encrypt only this tree. Unfortunately, the tree is very shallow and exhaustive search is easy to perform. In addition, we can often examine the coefficient matrix to determine the tree. It is not clear how partial encryption can be applied to wavelet packets algorithms.

Chapter 6

Partial Encryption of Video Sequences

The application of partial encryption to video communication is discussed in this chapter. The tremendous amount of data involved in video communication provides a strong motivation to reduce processing time for encryption and decryption. We will examine motion compensation and residual error coding separately. It is assumed that one of the partial encryption schemes for image compression introduced in Chapter 5 is used for intraframe coding.

6.1 Motion Compensation

The motion vectors must be encrypted in any secure video application. Otherwise, a cryptanalyst having access to an image frame may reconstruct good approximations to many successive frames, provided that the motion vectors are accurate. This is especially important if encryption is activated after the video transmission has started, so that the initial frames are available.

Typical motion compensation algorithms divide the current frame into blocks of fixed size, and compute a motion vector for each block. However, multiple blocks belonging to the same object may have identical or similar motion vectors, and it may be more efficient to code these motion vectors together. Schuster and Katsaggelos [34] proposed an algorithm that computes the motion vectors for fixed-sized blocks and

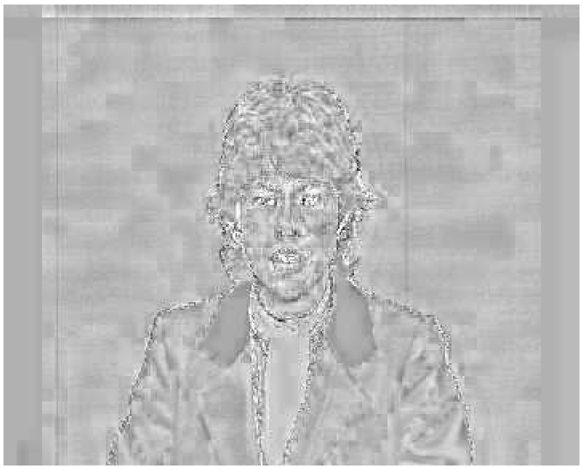


Figure 6.1: Residual error (after contrast enhancement) for frame 2 of the "claire" sequence.

merges the blocks using a quadtree. The advantage is that fewer motion vectors need to be transmitted and the overhead for transmitting the quadtree is low. The dimensions of the initial blocks are usually 8×8, so that an image frame of $2^n \times 2^n$ has a quadtree of maximum height $n - 3$.

Quadtree partial encryption for images can be adapted to encode motion vectors. The leaf values are the motion vectors instead of image intensities. As in the case of images, the quadtree decomposition is encrypted while the motion vectors are not. Instead of using a Hilbert curve to encode all the motion vectors at the same time as in [34], we encode the motion vectors one level at a time using Leaf Ordering II.

6.2 Residual Error Coding

The residual error often provides outlines of moving objects that are not perfectly predicted by the motion vectors, as shown in Figure 6.1. As a result, it is insufficient to encrypt only the intraframes and the motion vectors; the residual error must be encrypted to provide security. Since the residual error is simply an image frame, many video compression algorithms use standard image compression algorithms to compress the residual error. Both quadtree compression [34, 37, 39] and wavelet compression [24] have been used in video compression algorithms for residual error coding.

Strobach [37] suggested that quadtree compression is appropriate for residual error coding as the residual error often contains high intensities concentrated around the edges of objects. Wavelet compression may also be appropriate because of its excellent compression performance on still images. In Chapter 5, we have already discussed partial encryption schemes for both quadtree and wavelet image compression. These schemes can be applied directly to residual error coding.

6.3 File Sizes

The relative size of the quadtree of motion vectors can be predicted from the number of bits used to represent each motion vector. This is done in the same way as in the case of lossless quadtree image compression. For example, if the motion vectors belong to the set $\{(x, y) \in \mathbb{Z}^2 | -4 \leq x, y \leq 4\}$, there are 81 possible vectors and 7 bits are needed for each motion vector. In this case, the quadtree is at most 16.0% (Table 5.1) of the total output produced by the motion compensation step. Note that entropy coding may reduce the number of bits needed for each motion vector.

The relative size of the important part of the residual error is similar to the case of image compression. The results on the test video sequences are shown in Tables 6.1 and 6.2. In these experiments, motion compensation is performed on blocks of dimensions 8×8 initially. The relative sizes of the important parts in the quadtree partial encryption scheme are slightly larger than those in the case of still image compression. An explanation is that when the prediction by the motion vectors is accurate, most of the magnitudes of the residual error image are small. Therefore, the variance of the leaf values is small, so that the number of bits used to represent each leaf value is small. The size of the important part in SPIHT partial encryption is similar to the case of still image compression. As it is difficult to precisely control the bit rate of the quadtree compression algorithm, the results obtained for quadtree partial encryption and SPIHT partial encryption are not directly comparable.

Table 6.1: The size of important part of the residual error in the “football” sequence.

Frame	Quadtree		SPIHT	
	Bytes	%	Bytes	%
2	1808	28.8%	146	0.6%
3	2232	29.6%	140	0.5%
4	2124	29.0%	143	0.5%
5	2270	27.4%	152	0.6%
6	2425	27.8%	149	0.6%
7	2459	28.1%	144	0.5%
8	2473	24.0%	135	0.5%
9	2365	27.5%	133	0.5%
10	2646	27.9%	137	0.5%

Table 6.2: The size of important part of the residual error in the “claire” sequence.

Frame	Quadtree		SPIHT	
	Bytes	%	Bytes	%
2	90	27.0%	141	2.2%
3	162	26.0%	141	2.2%
4	201	26.7%	140	2.1%
5	190	28.0%	156	2.4%
6	212	26.9%	141	2.2%
7	211	28.5%	137	2.1%
8	226	28.8%	140	2.1%
9	224	30.7%	140	2.1%
10	256	28.0%	136	2.1%

6.4 Security

One property possessed by both the motion vectors and the residual error is that large homogeneous regions exist in both cases. The motion vectors for the blocks belonging to the same object tend to be identical or similar, and all static objects have the same motion vector—the zero vector. In the case of residual error, large regions of small magnitudes exist when the prediction by motion compensation is accurate.

It is important to ensure that the partial encryption schemes for both motion vectors coding and residual error coding are secure even if there are large homogeneous regions. In Chapter 5, we saw that large homogeneous regions may produce long runs in quadtree partial encryption if Leaf Ordering I is used. Consequently, it is critical that Leaf Ordering II is used in quadtree partial encryption of video sequences. In the case of SPIHT partial encryption on the residual error, cryptanalysis is not facilitated by large homogeneous regions. It is still difficult to correctly decode the image frame without the initial updates to the internal states of the algorithm. It is possible for the residual error to have very few large intensities, and this can be detected in both quadtree partial encryption and SPIHT partial encryption. However, it only indicates that the motion compensation is accurate; the current frame cannot be reconstructed without the motion vectors and the previous frame.

The security of the partial encryption schemes for video depends not only on the security of the corresponding schemes for images, but also on the inability of a cryptanalyst to take advantage of the temporal correlation among consecutive frames. Experiments have shown that although the motion vectors and the residual error are often predictable from those of the previous frames, much of the correlation among consecutive frames are lost in both the important part and the unimportant part. In the case of motion vectors coding, the quadtrees for consecutive frames are drastically different because quadtrees are highly sensitive to the placement of the moving objects [37], which is illustrated in Figure 6.2. Similarly, the quadtrees for residual

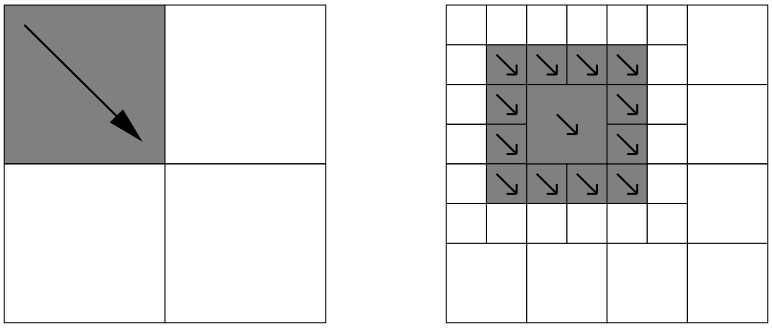


Figure 6.2: Sensitivity of quadtrees to the placement of a moving object.

coding differ significantly among consecutive frames. When SPIHT partial encryption is used for residual error coding, the algorithm produces drastically different output for consecutive frames because the zerotrees, like quadtrees, are highly sensitive to the placement of moving objects in the residual error.

The author feels that it is difficult to make use of temporal correlation in the cryptanalysis of partial encryption schemes. Nevertheless, it is possible that an attack exists. It remains an open question whether temporal correlation can be used to recover any information about the original video sequence.

6.5 Implementation Issues and Complexity

The overhead of separating the important part from the unimportant part is negligible as in the case of image compression. Since quadtree compression is used for encoding the motion vectors, a similar version can also be used to perform residual error coding if we wish to reduce implementation time or reduce program size. In addition, the encryption of the important part can be performed in parallel to the transmission of the unimportant part. Consequently, the encryption time may become negligible despite the large amount of data transmitted. However, the receiver cannot parallelize the decoding process because the important parts are needed before the unimportant parts.

6.6 Other Video Compression Algorithms

Many image compression algorithms can be extended to perform compression on three-dimensional signals. For example, quadtree compression algorithms can be extended to octree [10] algorithms, and two-dimensional wavelet compression algorithms can be extended to three-dimensional algorithms [20]. In both cases, it is straightforward to extend our partial encryption schemes for images to schemes for video sequences. We simply group multiple consecutive frames together and perform three-dimensional compression on the entire group.

There are also other video compression algorithms that have not been examined in this thesis. For instance, model-based algorithms [21] use a model to describe the content of each frame and transmit only the parameters of the model. For example, a model can be the human face, and the parameters may specify the movements of the various points on the face. These algorithms can achieve a tremendous amount of compression at the expense of lower video quality. Since the amount of data transmitted is small, it is not clear if partial encryption is useful in this case.

Chapter 7

Conclusion

We have examined several existing algorithms aimed at combining compression and encryption and found that they are inadequate. Instead, we proposed a partial encryption approach to reduce the encryption and decryption time in image and video communication. The output of an existing compression algorithm is decomposed into the important part and the unimportant part, and only the important part is encrypted. We showed how to perform secure partial encryption in image communication using quadtree compression and wavelet compression based on zerotrees, such that a relatively small proportion of the output is encrypted. We have also proposed extensions to video communication.

For image communication, typically 13%–27% of the total output is encrypted in quadtree partial encryption. We have also shown that it is difficult to cryptanalyze this scheme. On the other hand, less than 2% of the total output is encrypted for images of dimensions 512×512 in wavelet partial encryption based on zerotrees, and this partial encryption scheme also appears to be secure. The computational overhead in these schemes is negligible, and the compression performance of the chosen algorithms is not affected.

The encryption and decryption time for the tremendous amount of data involved in video communication is the main motivation for this work. We have shown that quadtree partial encryption and SPIHT partial encryption can easily be extended to video compression. Quadtree partial encryption can be used on the motion vectors, while either quadtree or SPIHT partial encryption may be used on the residual error. The computation overhead is also negligible, and the relative size of the important part is comparable to that in the case of image compression. Thus, the processing time for encryption and decryption is significantly reduced. Partial encryption is feasible because it can easily be implemented and it is computationally simple. We conclude that partial encryption is useful in making real-time secure image and video communication practical.

7.1 Limitations

The partial encryption approach has two major limitations. First, since the unimportant part is unencrypted, it is theoretically less secure than encrypting both the important part and the unimportant part. In practice, however, we have seen that it may still be computationally infeasible to cryptanalyze the partial encryption scheme. Moreover, a different partial encryption scheme has to be designed and analyzed for each type of compression algorithm. We do not have general techniques that work with all types of compression algorithms.

We have seen that if the quadtree is almost empty or almost complete, the number of possible quadtrees is small. Thus, quadtree partial encryption cannot be used on images that are too simple or too complex. In the first case, the image is highly compressible and little time is required for encryption or decryption, making partial encryption unnecessary. On the other hand, most of the images with almost complete quadtrees are likely filled with noise. As we have seen, natural images generally do not have almost complete quadtrees. For SPIHT partial encryption, cryptanalysis may be feasible for very simple images. However, very complex images do not facilitate the cryptanalysis of the partial encryption scheme.

7.2 Future Directions

Some of the possible directions to pursue in future research are listed below:

Region-based compression. Many other region-based compression algorithms also use tree structures to represent the locations and shapes of the regions. It may be possible to apply partial encryption to these algorithms.

Model-based compression. Model-based compression algorithms can be used to achieve a very high compression ratio. It is not clear whether there is a need for partial encryption, and if so, whether it is possible to apply partial encryption to these algorithms.

Human visual system modifications. Some researchers attempt to modify existing compression algorithms to incorporate the human visual system (HVS) into the design, so that the quality of the decoded images or video sequences is perceptually good. If HVS modifications are done to quadtree or SPIHT compression, it is not clear whether it is easier to cryptanalyze the corresponding partial encryption scheme. For example, HVS modifications may make one type of quadtrees occur with a higher probability.

General design and analysis techniques. It is useful to develop techniques or principles for the design and analysis of partial encryption schemes, so that we do not need to consider each type of compression algorithms separately.

Bibliography

[1] M. Bertilsson, E. F. Brickell, and I. Ingemarsson. Cryptanalysis of video encryption based on space-filling curves. In J.-J. Quisquater and J. Vandewalle, editors, *Advances in Cryptology—EUROCRYPT '89 Proceedings*, number 434 in Lecture Notes in Computer Science, pages 403–411. Springer-Verlag, 1990.

[2] N. Bourbakis and C. Alexopoulos. Picture data encryption using scan patterns. *Pattern Recognition*, 25(6):567–581, 1992.

[3] H. K.-C. Chang and J.-L. Liu. A linear quadtree compression scheme for image encryption. *Signal Processing: Image Communication*, 10(4):279–290, Sep 1997.

[4] H. Cheng and X. Li. On the application of image decomposition to image compression and encryption. In P. Horster, editor, *Communications and Multimedia Security II*, pages 116–127. International Federation for Information Processing, Chapman & Hall, 1996.

[5] R. J. Clarke. *Transform Coding of Images*. Academic Press, 1985.

[6] C. D. Creusere. A new method of robust image compression based on the embedded zerotree wavelet algorithm. *IEEE Transactions on Image Processing*, 6(10):1436–1442, Oct 1997.

[7] E. A. B. da Silva, D. G. Sampson, and M. Ghanbari. A successive approximation vector quantizer for wavelet transform image coding. *IEEE Transactions on Image Processing*, 5(2):299–310, Feb 1996.

[8] N. G. de Bruijn, D. E. Knuth, and S. O. Rice. The average height of planted plane trees. In R. C. Read, editor, *Graph Theory and Computing*, pages 15–22. Academic Press, 1972.

[9] R. Distasi, M. Nappi, and S. Vitulano. Image compression by B-tree triangular coding. *IEEE Transactions on Communications*, 45(9):1095–1100, Sep 1997.

[10] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, 2nd edition, 1990.

[11] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Addison-Wesley, 1992.

[12] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(2):1098–1101, Sep 1952.

[13] D. Jones. Data compression and encryption algorithms. World Wide Web. `http://www.cs.uiowa.edu/~jones/compress/`.

[14] D. Jones. Applications of splay trees to data compression. *Communications of the ACM*, pages 996–1007, Aug 1988.

[15] J. Knipe. A comparison of improved spatial and transform domain compression schemes. Master's thesis, University of Alberta, 1996.

[16] J. Knipe, X. Li, and B. Han. An improved lattice vector quantization based scheme for wavelet compression. *IEEE Transactions on Signal Processing*, 46(1):239–243, Jan 1998.

[17] D. E. Knuth. *The Art of Computer Programming: Fundamental Algorithms*, volume 1. Addison-Wesley, 3rd edition, 1997.

[18] X. Lai. *On the Design and Security of Block Ciphers*, volume 1 of *ETH Series in Information Processing*. Konstanz: Hartung-Gorre Verlag, 1992.

[19] X. Lai and J. L. Massey. A proposal for a new block encryption standard. In *Advances in Cryptology–EUROCRYPT '90 Proceedings*, number 473 in Lecture Notes in Computer Science, pages 389–404. Springer-Verlag, 1991.

[20] A. S. Lewis and G. Knowles. Video compression using 3D wavelet transforms. *Electronic Letters*, 26(6):396–398, Mar 1990.

[21] H. Li, A. Lundmark, and R. Forchheimer. Image sequence coding at very low bitrates: A review. *IEEE Transactions on Image Processing*, 3(5):589–609, Sep 1994.

[22] X. Li, J. Knipe, and H. Cheng. Image compression and encryption using tree structures. *Pattern Recognition Letters*, 18(11–13):1253–1259, Nov 1997.

[23] X. Liu, P. G. Farrell, and C. A. Boyd. Resisting the Bergen-Hogan attack on adaptive arithmetic coding. In M. Darnell, editor, *Cryptography and Coding: 6th IMA International Conference*, number 1355 in Lecture Notes in Computer Science, pages 199–208. Springer-Verlag, 1997.

[24] S. A. Martucci, I. Sodagar, T. Chiang, and Y.-Q. Zhang. A zerotree wavelet video coder. *IEEE Transactions on Circuits and Systems for Video Technology*, 7(1):109–118, Feb 1997.

[25] Y. Matias and A. Shamir. A video scrambling technique based on space filling curves. In C. Pomerance, editor, *Advances in Cryptology—CRYPTO '87 Proceedings*, number 293 in Lecture Notes in Computer Science, pages 398–417. Springer-Verlag, 1988.

[26] J. L. Mitchell, W. B. Pennebaker, C. E. Fogg, and D. J. LeGall. *MPEG Video: Compression Standard*. Chapman & Hall, 1996.

[27] J. Modayil, H. Cheng, and X. Li. An improved piecewise approximation algorithm for image compression. *To appear in Pattern Recognition*.

[28] J. Modayil, H. Cheng, and X. Li. Experiments in simple one-dimensional lossy image compression schemes. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems, June 3–6, 1997*, pages 614–615, 1997.

[29] W. B. Pennebaker and J. L. Mitchell. *JPEG Still Image Data Compression Standard*. Van Nostrand Reinhold, 1993.

[30] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digitial signatures and public key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[31] N. Roeder. Image compression and quality assessment: An investigation exploring wavelet packets and human visual system characteristics. Master's thesis, University of Alberta, 1997.

[32] A. Said and W. A. Pearlman. A new, fast, and efficient image codec based on set partitioning in hierarchical trees. *IEEE Transactions on Circuits and Systems for Video Technology*, 6(3):243–250, Jun 1996.

[33] B. Schneier. *Applied Cryptography: protocols, algorithms, and source code in C*. John Wiley & Sons, second edition, 1996.

[34] G. M. Schuster and A. K. Katsaggelos. *Rate-Distortion Based Video Compression: Optimal Video Frame Compression and Object Boundary Encoding*. Kluwer Academic Publishers, 1997.

[35] J. M. Shapiro. Embedded image coding using zerotrees of wavelet coefficients. *IEEE Transactions on Signal Processing*, 41(12):3445–3462, Dec 1993.

[36] E. Shusterman and M. Feder. Image compression via improved quadtree decomposition algorithms. *IEEE Transactions on Image Processing*, 3(2):207–215, Mar 1994.

[37] P. Strobach. Tree-structured scene adaptive coder. *IEEE Transactions on Communications*, 38(4):477–486, Apr 1990.

[38] P. Strobach. Quadtree-structured recursive plane decomposition coding of images. *IEEE Transactions on Signal Processing*, 39(6):1380–1397, Jun 1991.

[39] G. J. Sullivan and R. L. Baker. Efficient quadtree coding of images and video. *IEEE Transactions on Image Processing*, 3(3):327–331, May 1994.

[40] E. Walach and E. Karnin. A fractal based approach to image compression. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 1, pages 529–532, Apr 1986.

[41] I. H. Witten, R. Neal, and J. G. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30:520–540, Jun 1987.

[42] X. Wu. Image coding by adaptive tree-structured segmentation. *IEEE Transactions on Information Theory*, 38(6):1755–1767, Nov 1992.