Partial Encryption of Compressed Images and Videos[*]

Howard Cheng[†] and Xiaobo Li[‡]
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada T6G 2H1
Tel: (780) 492-2299 Fax: (780) 492-1071
E-mail: hchcheng@scg.math.uwaterloo.ca
li@cs.ualberta.ca

Abstract

The increased popularity of multimedia applications places a great demand on efficient data storage and transmission techniques. Network communication, especially over a wireless network, can easily be intercepted and must be protected from eavesdroppers. Unfortunately, encryption and decryption are slow and it is often difficult, if not impossible, to carry out real-time secure image and video communication and processing. Methods have been proposed to combine compression and encryption together to reduce the overall processing time [3, 4, 12, 18, 20], but they are either insecure or too computationally intensive.

We propose a novel 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 paper. Only 13%–27% of the output from quadtree compression algorithms [13, 17, 29, 30, 31, 32] is encrypted for typical images, and less than 2% is encrypted for 512×512 images compressed by the SPIHT algorithm [26]. The results are similar for video compression, 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.

EDICS Number: SP 7.8

[*]This research is supported in part by the Motorola Wireless Data Group and the Canadian Natural Sciences and Engineering Research Council under Grant OGP9198 and Postgraduate Scholarship.

[†]Presently at Department of Computer Science, University of Waterloo.

[‡]To whom correspondence should be addressed.

1 Introduction

The use of image and video applications such as the World Wide Web and video conferencing has increased dramatically in recent years. When communication bandwidth or storage is limited, data is often compressed. Especially when a wireless network is used, low bit rate compression algorithms are needed because of the limited bandwidth. On the other hand, encryption is also performed if it is necessary to protect the privacy of the users. For example, transmissions over a wireless network can easily be intercepted. 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 decoder.

The processing time for encryption and decryption is a major bottleneck in real-time image and video communication and processing. Moreover, 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. The computational overhead incurred by encryption and decryption algorithms make it impossible to handle the tremendous amount of data processed. For example, MPEG-2 compression produces output at a bit rate of 10 Mb/s or higher [21]. Compression algorithms for high-quality video sequences may generate even higher bit rates. In many cases, 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 makes real-time secure video communication and processing difficult, if not impossible. 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 processing time and computational requirement is important not only for these portable devices, but also 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 [20] 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. Chang and Liu [4] recently noted that it is still difficult to perform both compression and encryption quickly. Researchers have proposed methods to combine compression and encryption into a single process to reduce the total processing time [3, 4, 12, 18, 20], but these methods are insecure or too computationally intensive. To the best of the authors' knowledge, there has been no satisfactory solution proposed.

We propose a novel approach, called *partial encryption*, to reduce encryption and decryption time in image and video communication and processing. In this approach, only part of the compressed data is encrypted. It was shown in [5, 6] that this approach is not suitable for the JPEG [24] and MPEG [21] compression algorithms. Instead, we propose partial encryption schemes for quadtree and wavelet image compression, as well as an extension for video compression. Partial encryption allows the encryption and decryption time to be significantly reduced without affecting the compression performance of the underlying compression algorithm. It will also be shown that although a large portion of the compressed data is left unencrypted, it is difficult to recover the original data without decrypting the encrypted part. In the case of quadtree image compression [13, 17, 29, 30, 31, 32], the encrypted portion is 13%–27% of the compressed output for typical images. For wavelet compression based on zerotrees [8, 13, 14, 26, 28], less than 2% of the compressed output is encrypted for 512×512 images. Applications using low bit rate compression algorithms would result in smaller portions to be encrypted, making the partial encryption scheme even more feasible and attractive. The results on video compression is similar. Thus, a significant reduction in encryption and decryption time is achieved. In fact, the encrypted part may be so small that public-key encryption can be applied directly, so that the implementation and operating costs of secret-key encryption are eliminated. Moreover, there are other potential applications of partial encryption. Partial encryption was first applied to quadtree image compression by the authors in [6, 17], and there is no known cryptanalysis at this time.

The partial encryption approach and its potential applications are described in more detail in Section 2. In Section 3, quadtree image compression and wavelet compression based on zerotrees are briefly described to facilitate the discussion of the proposed schemes. In Section 4, the inadequacies of related algorithms are summarized. Partial encryption schemes for images are proposed and analyzed in Section 5, and an extension for videos is examined in Section 6.

2 Partial Encryption Based on Data Decomposition

Many compression algorithms for multimedia data decompose their input into a number of different logical parts. For example, region-based image and video compression produces the shapes and locations of the regions as well as the parameters describing the regions. Transform coding algorithms such as those given by the Joint Photographic Experts Group (JPEG) [24] and the Moving Picture Experts Group (MPEG) [21] produce coefficients corresponding to the chosen basis functions that represent components of different frequencies. Some of these algorithms have *important parts* that provide a significant amount of information about the original data, while the remaining parts may not provide much information without the important parts. For simplicity, we consider all important parts as one important part, and the remaining parts are grouped into one unimportant part. We use the term "information" loosely in this paper, 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 it provides a significant amount of information.

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 used to encrypt the important part. Figure 1 illustrates the difference between the proposed approach and the traditional approach, in which the entire output of the compression algorithm is encrypted. A significant reduction in encryption and decryption time is achieved when the relative size of the important part is small. Real-time secure video communication and processing are often impossible without the reduction. In some cases, partial encryption allows the important part to be encrypted while the unimportant part is transmitted in parallel, so that the encryption time becomes negligible. Moreover, partial encryption preserves a desirable property possessed by the traditional approach—the encryption and decryption time for highly compressible input sources remains low. Other related algorithms [3, 4, 12, 18, 20] do not possess this property.

In secure communication, a secret-key encryption algorithm such as IDEA [16] is typically used

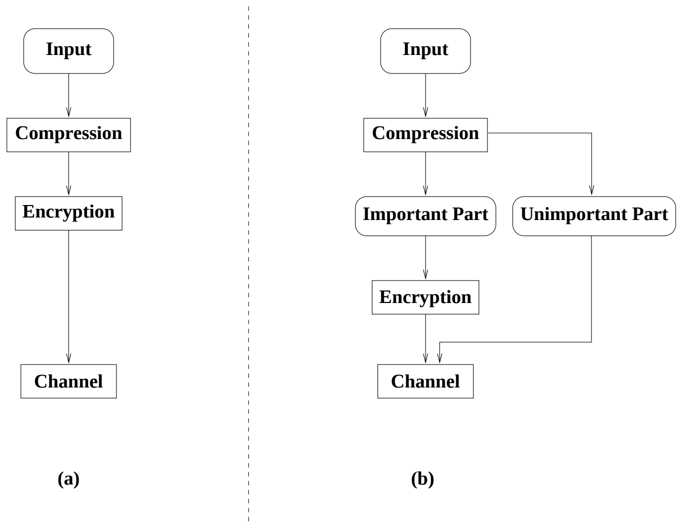

Figure 1: A comparison between (a) the traditional approach to secure image and video communication and (b) the proposed approach.

to encrypt the transmission, and the secret key is encrypted by a public-key encryption algorithm such as the RSA algorithm [25]. Public-key algorithms are used to solve the problem of key exchange, but they are too slow for encrypting a large amount of data. Consequently, they are not used to encrypt the actual message. In partial encryption schemes, the important part can be encrypted by a secret-key algorithm as described above. However, the important part may be so small that public-key algorithms can be applied directly to it, making secret-key encryption unnecessary. This is illustrated in Figure 2. The implementation and operation costs for secret-key encryption are completely eliminated in this case.

Partial encryption can also be viewed as a way to prioritize information, with the additional constraint that the unimportant part alone does not reveal much information about the original data. The important part often gives important visual information about an image as well. For example, we will show in Section 5 that the important part in quadtree compression algorithms shows the outlines of objects in an image. We may allocate more bandwidth for the important part or pay more attention to the important part in error-detection or error-correction coding, so that some visual information of the image can be obtained even if the unimportant part is totally lost. 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.

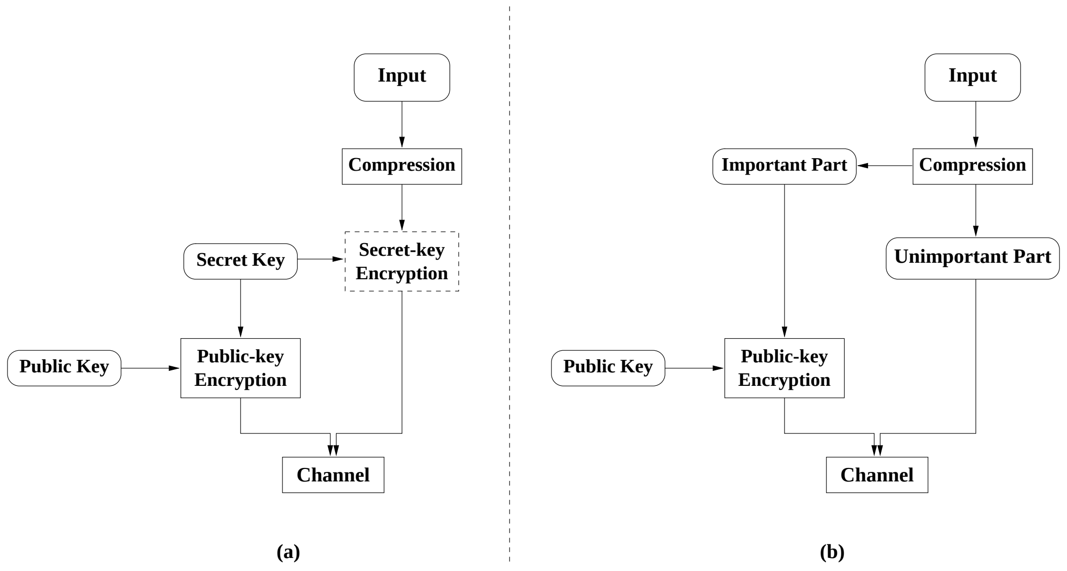

Figure 2: A comparison between (a) the traditional approach to secure image and video communication and (b) the proposed approach when public-key encryption can be applied directly to the important part.

Data decomposition and partial encryption have other applications such as distributed secure image and video retrieval and real-time secure broadcasting, respectively. In distributed image and video retrieval, the important parts are small and can be stored locally in a catalogue. When an image or a video is desired, only the unimportant part needs to be transmitted; hence, encryption is totally eliminated without compromising security. In real-time secure broadcasting, the transmission to each user may have to be encrypted with a different key to protect the privacy of the users. This "personalized" encryption is infeasible without the reduction in encryption time offered by partial encryption.

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 [6, 17], and no attacks have been found.

Finally, the proposed approach is a technique that can be applied to many tree-based 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.

3 Background

In this section, we briefly introduce cryptography, quadtree image compression, and wavelet image compression based on zerotrees to facilitate the discussion of the proposed partial encryption schemes. It is assumed that input images have been extended such that their dimensions are $2^n \times 2^n$ for some $n \geq 0$.

3.1 Cryptography

Cryptography is the study of keeping messages secure. The original message is called the *plaintext*, while the encrypted message is called the *ciphertext*. *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. Secure encryption algorithms must withstand the following 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 can be combined with any one of the three previous attacks to reduce the number of possible keys.

3.2 Quadtree Image Compression

Many variations of quadtree image compression algorithms exist [13, 17, 29, 30, 31, 32], and we only describe the basic concept here. Although more powerful compression algorithms exist, the computational complexity of quadtree compression is very low and it performs better than the JPEG algorithm at low bit rates [13, 17]. It is especially suitable for portable devices that may not have too much computing power.

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*. 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.

In lossless compression, 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. The lossy version is similar to the lossless counterpart, 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 [31].

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 implementa-

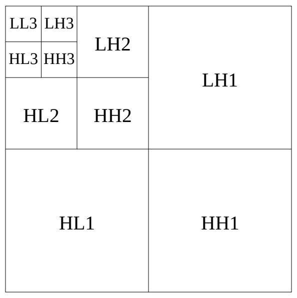

Figure 3: The hierarchy of wavelet coefficient bands.

tion, the algorithm starts with a complete quadtree of height n. 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.

3.3 Zerotree Wavelet Image Compression

The wavelet transform has been successfully applied to many image compression algorithms [1, 8, 13, 14, 26, 28]. It creates a hierarchy of coefficient bands, sometimes called a *pyramid decomposition*, as shown in Figure 3. The number in the band label is the *pyramid level* of the band. The LL band at the highest pyramid level is called the *root level*. It is assumed that wavelet transform is performed such that the root level has dimensions 8×8.

There is often correlation between coefficients that are in different pyramid levels of the hierarchy. Compression algorithms based on zerotrees [8, 13, 14, 26, 28] take advantage of this by grouping insignificant coefficients together into *zerotrees*, and indicate the insignificance of these coefficients very efficiently. We focus on the Set Partitioning in Hierarchical Trees (SPIHT) algorithm [26] in this paper because it is the basis of many other similar algorithms. In the SPIHT algorithm, each 2×2 block of coefficients in the root level corresponds to three trees of coefficients, as shown in Figure 4. The coefficient at (i, j) is denoted 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);

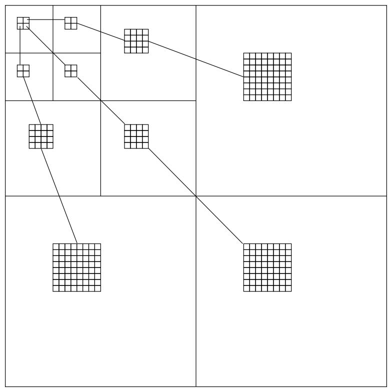

Figure 4: The trees of wavelet coefficients.

- $\mathcal{H}$: 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 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, representing the sets $\mathcal{D}(i,j)$ and $\mathcal{L}(i,j)$. 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. Initially, the LIP contains the coefficients in $\mathcal{H}$, the LIS contains $\mathcal{D}(i,j)$ entries where (i,j) are coordinates with descendants in $\mathcal{H}$, and LSP is empty. During the sorting pass, the significant coefficients in the LIS are identified by partitioning the sets $\mathcal{D}(i,j)$ into $\mathcal{L}(i,j)$ and the individual coefficients in $\mathcal{O}(i,j)$, or $\mathcal{L}(i,j)$ into $\mathcal{D}(k,l)$ where $(k,l) \in \mathcal{O}(i,j)$. Each significant coefficient is moved to the LSP. During the refinement pass, 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. A complete description of the algorithm can be found in [26].

4 Related Algorithms

Related algorithms have been proposed by other authors to combine compression and encryption [3, 4, 12, 18, 20]. The inadequacies of these algorithms are summarized here, and a more detailed analysis of these algorithms can be found in [5].

Jones [12] observed that in adaptive Huffman coding [10] and arithmetic coding [34], 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. He suggested that an adaptive compression algorithm can be used as an encryption algorithm if the initial model is kept secret. The key is a string of symbols that is encoded before the encoding or decoding of the actual message. Thus, the initial model is determined by the symbols in the key. On Jones' World Wide Web home page [11], it is stated that this algorithm is vulnerable to chosen-plaintext attacks—there are sequences of input symbols that can expose the current state of the model.

Liu *et al.* [18] recently proposed an algorithm that combines encryption and adaptive arithmetic coding. In addition to concealing the initial parameters of the algorithm as in Jones' proposal, it also performs secret adjustments to the current interval after each symbol is coded. 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***. Two pairs of multipliers are available, and the choice of the pair depends on a separate secret bit string. The authors claimed that the algorithm has many desirable properties for security, such as diffusion and the randomness of output. However, their experimental results show that the algorithm is approximately twice as slow as the original arithmetic coding algorithm by Witten *et al.* [34], and the compression performance is 2% worse than that of the original algorithm. Note that the extra operations required by the secret adjustments are applied for every symbol encoded, so the overhead is high even for an input source with low entropy.

Matias and Shamir [20] proposed a video encryption algorithm in which each frame is scrambled by a secret space-filling curve. Algorithms are available for compressing the scrambled signal [22, 23, 33]. Bertilsson *et al.* [2] gave a ciphertext-only attack that takes advantage of the temporal correlation among consecutive frames. If a different space-filling curve is used for each frame and there is little or no motion in the frames, the scrambled output from the different space-filling curves can be used to recover many pieces of the frames. In addition, this algorithm is simply a transposition cipher, and so it is vulnerable to known-plaintext and chosen-plaintext attacks [5].

Bourbakis and Alexopoulos [3] proposed another image encryption algorithm that scrambles an image using hierarchical scan patterns defined by the SCAN language. A substitution is performed on each pixel based on an additive noise vector that is part of the key. Compression may be performed by techniques such as run-length encoding. The number of distinct SCAN patterns is small and substitutions must be performed to withstand attacks by exhaustive key search [5]. For example, the number of SCAN patterns for an image of dimensions 512×512 is only 6.617867×10^{10} [3], which is less than 2^{36}. Furthermore, the substitutions can be completely determined by a chosen-plaintext attack using only one encryption [5]. The same chosen-plaintext attack may be used to determine the hierarchical decomposition specified by the key.

Chang and Liu [4] recently proposed an image compression and encryption algorithm based on lossless quadtree image compression. A permutation, or scanning order, is applied to the four branches at each internal node of the quadtree. The encryption key specifies the scanning order at each node. This algorithm does not appear to be practical because the length of the key is $\lceil \log_2(24)(4^n - 1)/3 \rceil$ bits for an image of dimensions $2^n \times 2^n$, so that the key length is linear in the size of the image. In addition, numerous attacks on this algorithm are available [5]:

Key space reduction. Many keys produce identical output on the same input image. The effective key space is dramatically reduced.

Histogram attacks. The image histogram of the original image is preserved in the output, revealing the average intensity of the image. This can be used to obtain a reconstruction of resolution $2^k \times 2^k$ by determining only the scanning orders used at the first k levels.

Known-plaintext attack. Pixels in the same quadrant in the original image are also in the same quadrant in the output. A signature that is invariant under the scanning orders used can be computed for each subtree, and the correspondence between the quadrants in the original image and the encrypted image can be determined. Consequently, the number of possible keys is reduced dramatically.

Chosen-plaintext attack. A very efficient chosen-plaintext attack can be performed by generating input images in which as many subtrees as possible have unique signatures. By applying the known-plaintext attack, the encryption key for 8-bit images of dimensions 512×512 can be recovered using only three encryptions.

The related algorithms described above are either insecure or too computationally intensive. We conclude that none of them satisfactorily solves the problem of reducing the overall compression and encryption processing time.

5 Partial Encryption of Images

Partial encryption of images is examined in this section. This approach cannot be blindly applied to any compression algorithm, and each proposed scheme must be evaluated carefully. For example, it was shown in [5, 6] that encrypting only low-frequency coefficients in transform-based compression such as JPEG [24] is inappropriate for partial encryption. The encrypted part is more than 50% of the total size of the compressed image, and outlines of objects are revealed.

We have found two classes of algorithms that are suitable for partial encryption. Quadtree compression algorithms [13, 17, 29, 30, 31, 32] are computationally simple and outperform JPEG at low bit rates [13, 17]. Wavelet compression algorithms based on zerotrees [8, 13, 14, 26, 28] have good compression performance. Both types of algorithms are suitable for low bit rate applications, and partial encryption schemes for them are proposed. The relative size of the important part, the computational complexity, and the security of each scheme are then analyzed. The size of the important part compared to the total size of the compressed image is directly proportional to the amount of encryption and decryption time required. Experimental results are obtained from an extensive set of test images that are commonly used by other researchers.

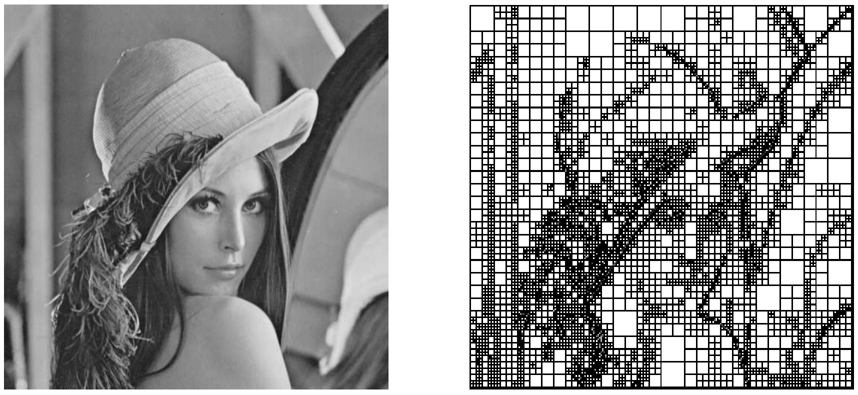
(a) Original image (b) Quadtree decomposition

Figure 5: Quadtree decomposition of an image.

5.1 Quadtree Compression

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 such as texture information or parameters for a first-order model [31]. The quadtree decomposition provides outlines of objects in the original image, as illustrated in Figure 5. 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. 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.

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. We concentrate on the bit allocation used by Shusterman and Feder [29]:

$$b_i = \frac{1}{2}\log\left(\frac{\sigma_i^2 L}{4^i D}\right), \qquad (1)$$

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 is the total number of leaf nodes in the quadtree (denoted L_{qt} in [29]), and

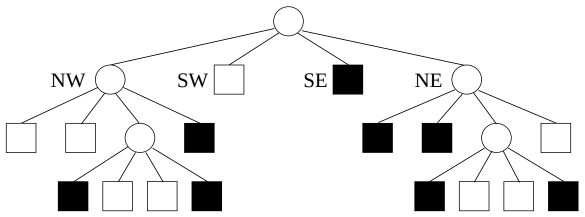

Figure 6: An example quadtree for a binary image.

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. Since the bit allocation can be represented by very few bits, it is also included in the encrypted part.

The leaf values must be transmitted in some order, and two orderings of the leaf values are introduced. The quadtree in Figure 6 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.

The first leaf values ordering, Leaf Ordering I, is induced by the inorder traversal of the quadtree [15]. In this ordering, the leaf values are encoded as 0010011011110010. In Leaf Ordering II, the leaf values are encoded one level at a time from the highest level to the lowest level, since this ordering is natural for bottom-up quadtree construction. This is opposite to breadth-first traversal of the tree, in which leaf values are concatenated from the lowest level to the highest level. At each level, the blocks corresponding to the leaf values are ordered by the raster scan—each row of blocks is 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. The leaf values of the levels are concatenated, and no special encoding is needed to separate each level if the quadtree is also known. Thus, the final encoded output is 1111000001100101. A more detailed example can be found in [5].

It will be shown that certain properties possessed by Leaf Ordering I make it susceptible to cryptanalysis. These properties are not present in Leaf Ordering II, making it much more difficult to cryptanalyze. Leaf Ordering I is not secure; it is presented to illustrate that the structure of the unimportant part must be considered in designing partial encryption schemes.

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 [15]. The proofs are omitted here for brevity.

Lemma 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.*

Theorem 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.*

Corollary 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 [9].

Theorem 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,*

$$g_h(z) = \begin{cases} 1+z & \text{if } h = 0, \\ 1+z(g_{h-1}(z))^4 & \text{if } h > 0. \end{cases} \tag{2}$$

Proof. The theorem follows directly from Corollary 3 and induction on h. □

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. To simplify the calculations, we assume that the quadtree is represented by one bit for each node to obtain upper bounds on the quadtree size. By Lemma 1, 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 (i.e., each pixel may assume one of 2^b possible intensity levels), the total size of the leaf values is $b(3k+1)$ bits. It follows that an approximate upper bound on the relative quadtree size, defined as the ratio of the size of the quadtree and the total size of the compressed image, is

$$\frac{4k+1}{4k+1+b(3k+1)} = \frac{4+\frac{1}{k}}{3b+4+\frac{b+1}{k}} \approx \frac{4}{3b+4},$$

or $(400/(3b+4))\%$. The approximation is valid since k is typically at least 1000 for 256×256 images, and greater for larger images. Table 1 shows the upper bounds for various values of b.

Table 1: Approximate upper bounds on the relative quadtree size in lossless quadtree compression on b-bit images.

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

For lossy compression, the situation is more complicated because the number of bits used for each leaf value is different. We see from (1) that the relative quadtree size is small if there are many leaf nodes at low levels. 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. Table 2 shows the results obtained from test images. The results show that the relative quadtree size 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%. The reason is that "washsat" is a low-contrast image, so that the values σ_i^2 in (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.

The analysis is similar when other parameters are used to describe the blocks associated with the leaf nodes. For example, Strobach [31] 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 was shown in [31] that at approximately 0.5 bpp, the quadtree is 11.7% for the "boat" image, and 8.4% for the "lena" image.

Quadtree partial encryption is secure against attacks by tree enumeration, in which the cryptanalyst generates all possible quadtrees and matches them against the unencrypted leaf values. For an image of dimensions $2^n \times 2^n$, the maximum height of the quadtree is n. In the case of lossless compression, 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. If the number of leaf nodes is L, 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 4, and are plotted for $h = 4, 5, 6$ in Figure 7. Unless k is very small or very large, the number of possible quadtrees is very large even for low-resolution images. As a result, exhaustive tree enumeration is infeasible unless the quadtree is almost empty or almost complete. This does not occur for typical images [5].

Table 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%

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.

A *run*, or a sequence of identical leaf values, in the encoded data can be used in 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. In Figure 8, a

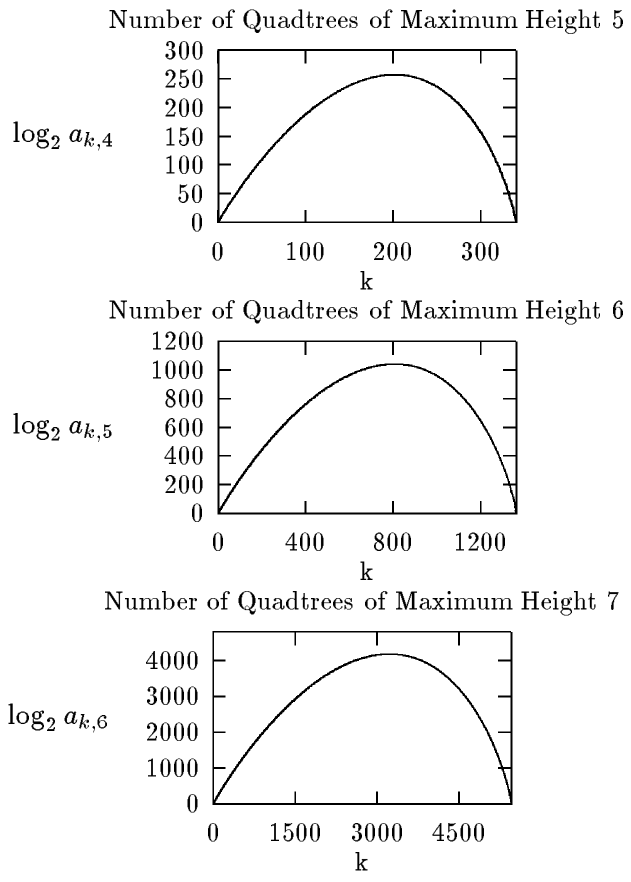

Figure 7: Plots of $\log_2 a_{k,h}$ for various values of h.

quadtree with a run of length 14 is shown. Notice that the levels of the leaf nodes in this run do not change direction (decreasing or increasing) many times. This follows from the fact that once the levels start increasing, they cannot decrease until four sibling leaf nodes are encountered in the run, which is impossible. As a result, long runs reduce the number of quadtrees that need to be examined in an exhaustive search.

Moreover, a homogeneous region can be recovered from a long run that starts at a NW leaf node. Let k be the length of the run starting at a NW leaf node of some subtree, and r be the leaf value in this run. A homogeneous region can 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

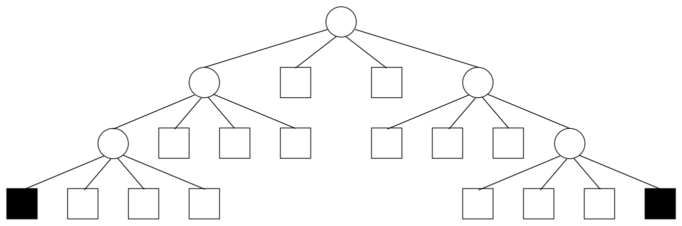

Figure 8: A quadtree with a run of length 14.

(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. It may be difficult to classify the leaf nodes corresponding to the start and the end of a run in general. However, the above procedure can be applied to the runs at the start (end) of the encoded leaf values, since the run starts at a NW leaf node (ends at a NE leaf node). This procedure can also be used in a known-plaintext attack. If a known non-homogeneous image portion is encoded as a subtree of the quadtree for the original image, quadtree compression can be applied to this portion to obtain the leaf values, which forms a substring of the leaf values of the entire image. The above procedure can then be used on 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. 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 8. 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. Long runs may be used to obtain homogeneous regions at the corners and may reveal the shapes of foreground objects.

In the case of lossy compression using Leaf Ordering I, it is crucial that the individual leaf values cannot be obtained from the concatenated bit stream without the quadtree. If the non-zero bit allocation for each level is distinct and the individual leaf values are obtained, we can determine the level at which each leaf value is located. As a result, the quadtree can be completely determined [5]. 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 as well.

Leaf Ordering II is 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. 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, and the number of possible quadtrees is large even with this extra information [5].

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. Thus, the computational overhead of using quadtree partial encryption is negligible.

5.2 Zerotrees Wavelet Compression

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 [26] transmits the significance of the coefficient sets that correspond to trees of coefficients. This is similar to quadtree compression as it indicates whether a set needs to be decomposed further. Instead of homogeneity, significance is the factor for deciding whether a set is partitioned. We focus on the SPIHT compression algorithm in the discussion, but other algorithms based on zerotrees can also be used [8, 13, 14, 28].

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 [7, 26]. 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. 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 $0 \leq i,j < 16$. 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.

The following result is proved in [5]:

Table 3: 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%

Theorem 5 *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. Furthermore, the size of the important part is at least 160 bits if at least two sorting passes are performed.*

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 3 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.

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, 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-quality approximation, but experiments show that at least 160 bits of the important part must be known to give a very coarse approximation. Figure 9 shows the experimental results on the "lena" image. Furthermore, it is difficult to distinguish the correct important part from the incorrect ones when the original image is 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. 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 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.

5. 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.

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.

(a) Original image

(b) $k = 100$

(c) $k = 120$

(d) $k = 140$

(e) $k = 160$

(f) $k = 180$

Figure 9: Reconstructed images using only the first k encrypted bits.

6 Partial Encryption of Videos

We now give extensions of the partial encryption schemes for videos. Video compression algorithms generally consist of two parts—motion compensation and residual error coding, which will be examined separately. It is assumed that one of the partial encryption schemes for images introduced in Section 5 is used for intraframe coding. We have already seen in Section 5 that quadtree-based partial encryption is secure against tree enumeration attacks even when the images have low resolution, and that the image resolutions has little effect on the security of zerotree-based partial encryption schemes. As a result, both schemes are suitable for low-resolution video applications such as "videophones."

The motion vectors must be encrypted. Otherwise, an image frame may be used to provide approximations to successive frames. This is especially important if encryption is activated after the video transmission has started, so that the initial frames are available to the cryptanalysts. 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 [27] proposed an algorithm that computes the motion vectors for fixed-sized blocks and merges the blocks using a quadtree. 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 the motion vectors as in [27], we encode the motion vectors one level at a time using Leaf Ordering II. Although the maximum height of the quadtree is reduced, we have seen in Section 5 that the number of possible quadtrees remains large for videos having medium or high resolutions. For low-resolution videos, the maximum height of the quadtree may be small enough to make exhaustive tree enumeration feasible. In that case, the motion vectors are completely encrypted.

The residual error often provides outlines of moving objects that are not perfectly predicted by the motion vectors, as shown in Figure 10. As a result, it is insufficient to encrypt only the intraframes and the motion vectors; the residual error must also be encrypted to provide security.

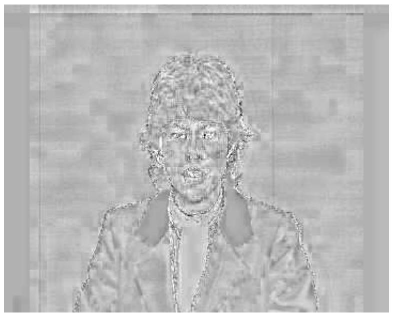

Figure 10: Residual error (after contrast enhancement) for frame 2 of the "claire" sequence.

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 [27, 30, 32] and wavelet compression [19] have been used in video compression algorithms for residual error coding. Strobach [30] 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 is also appropriate because of its excellent compression performance on images. Partial encryption schemes for both quadtree and wavelet image compression can be applied directly to residual error coding.

The relative size of the quadtree of motion vectors can be predicted from the number of bits used to represent each motion vector. For example, if each motion vector is represented by 6 bits, the quadtree is at most 18.2% (Table 1) of the total output produced by the motion compensation step. The relative size of the important part of the residual error is similar to the case of image compression. The results on two test video sequences are shown in Table 4. Each frame in the "football" sequence has dimensions 720×496, and each frame in the "claire" sequence has dimensions 360×288. In these experiments, motion compensation is performed on blocks of dimensions 8×8 initially. Only the results for the first ten frames of each video are shown for each video as the results are similar for other frames. 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. This occurs because most of the magnitudes of the residual error image are small when the prediction by the motion vectors is accurate. 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. Since it is difficult to precisely

Table 4: The size of important part of the residual error for the test video sequences.

	football				claire			
	Quadtree		SPIHT		Quadtree		SPIHT	
Frame	Bytes	%	Bytes	%	Bytes	%	Bytes	%
2	1808	28.8%	146	0.6%	90	27.0%	141	2.2%
3	2232	29.6%	140	0.5%	162	26.0%	141	2.2%
4	2124	29.0%	143	0.5%	201	26.7%	140	2.1%
5	2270	27.4%	152	0.6%	190	28.0%	156	2.4%
6	2425	27.8%	149	0.6%	212	26.9%	141	2.2%
7	2459	28.1%	144	0.5%	211	28.5%	137	2.1%
8	2473	24.0%	135	0.5%	226	28.8%	140	2.1%
9	2365	27.5%	133	0.5%	224	30.7%	140	2.1%
10	2646	27.9%	137	0.5%	256	28.0%	136	2.1%

control the bit rate of the quadtree compression algorithm, the results obtained for quadtree partial encryption and SPIHT partial encryption are not directly comparable.

Large homogeneous regions exist in both the motion vectors and the residual error. 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 the 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 Section 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 videos. 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, this implies that the motion compensation is accurate; the current frame cannot be reconstructed without the motion vectors and the previous frame. Note that a cryptanalyst cannot infer from this information that there is little or no motion in the video. It is possible for motion compensation to be very successful even when there is a large amount of motion in the video.

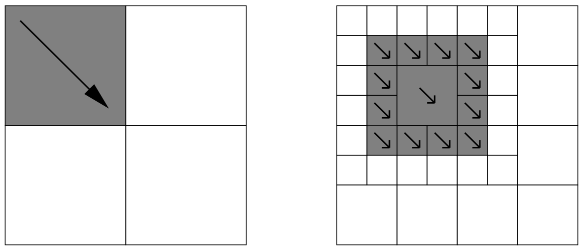

Figure 11: Sensitivity of quadtrees to the placement of a moving object.

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 is lost in both the important part and the unimportant part. In the case of motion vectors coding, the quadtrees of consecutive frames are drastically different because they are highly sensitive to the placement of the moving objects [30], which is illustrated in Figure 11. Similarly, the quadtrees for residual coding differ significantly among consecutive frames. When SPIHT partial encryption is used for residual error coding, the algorithm produces drastically different outputs for consecutive frames because the zerotrees, like quadtrees, are highly sensitive to the placement of moving objects in the residual error.

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 program size. Furthermore, 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.

7 Conclusion

In this paper, we proposed an approach, called partial encryption, to reduce the encryption and decryption time in image and video communication and processing. For both images and videos, the processing time for encryption and decryption is significantly reduced. Partial encryption is feasible because it can easily be implemented and is computationally simple. We conclude that partial encryption is useful in reducing the encryption and decryption time in secure image and video communication and processing.

The main limitation of our approach is that a different scheme has to be designed and analyzed for each compression algorithm. One possibility for future research is to identify the key elements of partial encryption to obtain general techniques for the design and analysis of partial encryption schemes. It appears that the important parts are often those that "drive" the decompression algorithm. In other words, the branch of execution of the decompression algorithm depends on the important parts. It is perhaps useful to examine important parts from this perspective. Another possibility is to examine the applicability of partial encryption to other region-based compression algorithms that also use tree structures to store the locations and shapes of the regions.

References

[1] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies. Image coding using wavelet transform. *IEEE Trans. Image Processing*, 1(2):205–220, Apr 1992.

[2] M. Bertilsson, E. F. Brickell, and I. Ingemarsson. Cryptanalysis of video encryption based on space-filling curves. In *EUROCRYPT '89*, pages 403–411, 1990.

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

[4] 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.

[5] H. Cheng. Partial encryption for image and video communication. Master's thesis, Univ. of Alberta, 1998.

[6] H. Cheng and X. Li. On the application of image decomposition to image compression and encryption. In *Communications and Multimedia Security II*, pages 116–127, 1996.

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

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

[9] 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.

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

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

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

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

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

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

[16] X. Lai. *On the Design and Security of Block Ciphers*, volume 1. Konstanz: Hartung-Gorre Verlag, 1992.

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

[18] X. Liu, P. G. Farrell, and C. A. Boyd. Resisting the Bergen-Hogan attack on adaptive arithmetic coding. In *Cryptography and Coding: 6th IMA Intl. Conf.*, pages 199–208, 1997.

[19] S. A. Martucci, I. Sodagar, T. Chiang, and Y.-Q. Zhang. A zerotree wavelet video coder. *IEEE Trans. Circ. and Syst. for Video Technol.*, 7(1):109–118, Feb 1997.

[20] Y. Matias and A. Shamir. A video scrambling technique based on space filling curves. In *CRYPTO '87*, pages 398–417, 1988.

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

[22] J. Modayil, H. Cheng, and X. Li. Experiments in simple one-dimensional lossy image compression schemes. In *Proc. IEEE Intl. Conf. on Multimedia Comp. and Syst.*, pages 614–615, 1997.

[23] J. Modayil, H. Cheng, and X. Li. An improved piecewise approximation algorithm for image compression. *Pattern Recognition*, 31(8):1179–1190, Aug 1998.

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

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

[26] A. Said and W. A. Pearlman. A new, fast, and efficient image codec based on set partitioning in hierarchical trees. *IEEE Trans. Circ. and Syst. for Video Technol.*, 6(3):243–250, Jun 1996.

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

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

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

[30] P. Strobach. Tree-structured scene adaptive coder. *IEEE Trans. Commun.*, 38(4):477–486, Apr 1990.

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

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

[33] E. Walach and E. Karnin. A fractal based approach to image compression. In *IEEE Intl. Conf. on Acoust., Speech Signal Processing*, volume 1, pages 529–532, Apr 1986.

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

List of Figures

List of Tables