JOINT COMPRESSION AND ENCRYPTION

A Thesis
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Ahana Ghosh

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Electrical and Computer Engineering

May 2010

Fargo, North Dakota

North Dakota State University
Graduate School

Title

JOINT COMPRESSION AND ENCRYPTION

By

Ahana Ghosh

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

ABSTRACT

Ghosh, Ahana, M.S.,Department of Electrical and Computer Engineering, College of Engineering and Architecture, North Dakota State University, May 2010. Joint Compression and Encryption. Major Professor: Dr. Rajendra Katti.

This research work proposes two techniques of joint compression and encryption using Shanon Fano Elias coding and Arithmetic coding respectively. The first scheme proposed is called Adaptive Shannon-Fano-Elias code where it has been observed that the complexity of attacks is exponential in m, where m is the length of the string being compressed. Since m is usually very large ($> 2^{20}$), the security of our scheme is very high. The main reason why our scheme's security depends on m is the fact that all attacks require the ciphertext to be scanned from left to right. The algorithm proposed does not compromise in the compression ratio produced by normal Shannon Fano Elia coding. The second scheme proposed uses Arithmetic coding as the compression algorithm as Arithmetic coding is one of the optimal compression techniques that can be used in various applications. The algorithm proposed is proved to be secure under an assumption that the attacker would have access to an algorithm which could decrypt any given message without the knowledge of the key.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

LIST OF FIGURES continued

Figure Page

CHAPTER 1. INTRODUCTION

1.1. Introduction

In the last decade, the explosion of information in our daily lives has made data compression an important requirement in communication. Extensive research is done in developing algorithms and techniques to solve the problem of storing and transmitting large data files. The basic idea of compression is to identify the structure of data and represent it in compact form. A number of existing compression algorithms, such as Huffman coding, Shannon Fano Elias (SFE) coding, and Arithmetic coding, are used as compression techniques in today's multimedia and communication systems. Another factor which has gained importance for existing communication systems is security. Compression does not assure security; therefore, the compressed messages need to be encrypted before sending them to the other end of the communication channel. One of the interesting methods of implementing security and compression together is to use any of the existing compression algorithms for encryption. The dual use of such compression algorithms saves time and effort in embedded multimedia systems [1].

Mohtashami presented the cryptographic aspect of Huffman coding in his paper [2]. However, if the probability mass function (pmf) of the source is known, Huffman coding does not act as one of the efficient encryption algorithms. Another disadvantageous factor with Huffman coding is large computations. To overcome such disadvantages, methods have been proposed where SFE coding is used for both encryption and decryption [3]. Unlike Huffman, coding SFE coding is proven to be secure for the sources whose pmfs are known. Another strong compression technique which can be used for encryption is

Arithmetic coding. Arithmetic coding uses a model-based paradigm where an encoded string can be produced from an input string of symbols and a pre-defined model. This encoded string is a compressed version of the input string. There are few limitations associated with both SFE and Arithmetic coding. SFE coding does not produce optimum compression. On the other hand, Arithmetic coding is one of the strongest compression techniques but it involves a lot of computation which requires both time and memory. This thesis work addresses the problem of joint compression and encryption. It proposes two new methods where SFE and Arithmetic coding are used. The proposed methods are cryptographically more secure than the existing standard algorithms. Cryptanalysis of these new methods has also been presented in this thesis. Before we provide the details of the new methods, a background of the existing compression techniques and their advantages/disadvantages in use as an encryption algorithm is discussed.

In Section 1.2, coding principle and features of Huffman coding are discussed in detail. Section 1.3 talks about Shannon Fano Elias Coding, whereas Section 1.4 gives a detailed study of Arithmetic Coding.

1.2. Huffman Coding

Huffman coding is a technique introduced by David A. Huffman in 1952.

It is one of the best-known compression techniques, producing optimal compression for any given probability distribution. Let a source S be putting out symbols $S = s_1, s_2,\ldots, s_m$ where $s_i \in \mathcal{X} = \{x_1,x_2,\ldots,x_n\}$. Set $\mathcal{X}$ is called the source alphabet. Let us assume that the pmf of $\mathcal{X}$ is $p(x_i) > 0$. In Huffman coding, the encoder needs to arrange these probabilities [$p(x_i)$ s] in ascending order before finding the codeword for each symbol. Ordering of symbols is needed at each step while encoding a string of symbols [4]. Huffman codewords are

optimal; i.e., no other codeword could have an expected length lower than these codewords. The expected length of a code is defined as follows:

$$L = \sum_{i=1}^{n} p(x_i) l(x_i) , \qquad (1.1)$$

where $l(x_i)$ is the length of the codeword of x_i having a probability of $p(x_i)$. It can be shown that L satisfies $H<=L<H+1$, where H represents entropy of the source [4].

One of the disadvantages of Huffman coding is that it involves large computations. This may lead to the addition of extra hardware during implementation of Huffman coding. Moreover, if Huffman coding is used for encryption, the probability mass function of the source must be hidden. This is because, by knowing the probability mass function of the source, it is very easy for an attacker to find the Huffman codewords for all the symbols in the symbol set. On the other hand, it has been shown in [2] that, if an attacker is not aware of the pmf of the source, then a Huffman coded file becomes quite difficult to crypatanalyze. However, in most cases like plain English text, the pmf of the source is known. Therefore, in spite of having good compression capability, Huffman coding is not a very good technique to use for encryption. Suppose we have the following symbols in a symbol set. The respective probabilities are tabulated in Table 1.

Table 1. Symbols showing their respective probabilities

Symbols	Probabilities
A	0.5
B	0.2
C	0.2
D	0.1

Using Huffman coding, the following code tree (Figure 1.1) could be drawn where the symbols with low probabilities are placed at the bottom of the tree while those of high probabilities are on the top. All the left branches of the tree are assigned the bit '0', and those on the right are assigned '1'. Starting from node 1 and tracing the path in the code tree, the codewords of the symbols are determined as shown in Table 2.

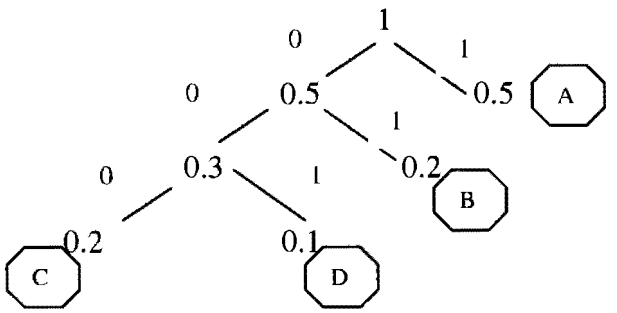

Figure 1.1. Code Tree for Huffman coding.

Table 2. Codeword for the symbols shown in Table 1.

Symbol	Probabilities	Codeword
A	0.5	1
B	0.2	01
C	0.2	000
D	0.1	001

1.3. Shannon Fano Elias Coding

Shannon Fano Elias coding is one of the compression algorithms which has stronger encryption capability than Huffman coding. SFE coding does not need to follow any particular ordering of symbols to find the codewords. SFE coding was first introduced in a 1963 information book by Abramson [1]. The construction of SFE coding is based on finding the modified cumulative frequency of all symbols in the symbol set. Let us consider the source alphabet, $\mathcal{X}$, as defined in Section 1.2. The modified cumulative distribution function is defined as

$$\bar{F}(x_i) = \sum_{k=1}^{i-1} p(x_k) + \frac{1}{2} p(x_i) \, . \tag{1.2}$$

$\bar{F}(.)$ represents the sum of the probabilities of all symbols placed before x_i in the ordering plus half of the probability of x_i. Because the random variable is discrete, the cumulative function consists of a step size of $p(x_i)$. $\bar{F}(x_i)$ is actually a real number only expressible by an infinite number of bits. Therefore, $\bar{F}(x_i)$ can be rounded off to $l(x_i) = \left\lceil \log \frac{1}{p(x_i)} \right\rceil + 1$ bits, denoted by $\lfloor \bar{F}(x_i) \rfloor_{l(xi)}$. SFE coding uses $\lfloor \bar{F}(x_i) \rfloor_{l(xi)}$ as the codeword for x_i. The resulting codewords are proven to be prefix-free [2][5]. The expected length, L, of an SFE-encoded sequence is defined as

$$L = \sum_{i=1}^{n} p(x_i) l(x_i) \, . \tag{1.3}$$

It can be shown that L satisfies $H+1<=L<H+2$, where H represents entropy of the source [2]. It is implied from equation (2) that the modified cumulative frequency of x_i depends on the probabilities of the symbols preceding x_i in that particular order of the symbols in the symbol set. Therefore, if the order is changed, $\bar{F}(x_i)$ and, subsequently $\lfloor \bar{F}(x_i) \rfloor_{l(xi)}$, would change. Different orderings would result in different codewords for a particular

symbol. This implies that the order of the symbols can be used as the key during encryption. Cryptanalysis of such an algorithm is shown in [3]. Unlike Huffman coding, SFE can be used for a system where pmf of the source is known, but SFE does not produce good compression because codewords produced by SFE contains redundant bits. We can get rid of such redundant bits by applying the algorithm proposed in [6].

In this thesis, we propose a new method of encryption using SFE. Irrespective of source pmf, the algorithm assumes the initial distribution to be uniform. After every t symbols (where t is a constant), the pmf is changed. Hence, the method is named as Adaptive SFE. It is also shown that this t is proportional to m i.e., the length of the message, and, therefore, is a very large number.

Example of SFE Coding

Let a source alphabet $X = \{a_1, a_2, a_3\}$. The probabilities and the corresponding modified cumulative frequency are shown in Table 3. The codewords are the binary representation of the modified cumulative frequency. $\bar{F}(x_i)$ is found following equation (1.2). Therefore, if we have a message, $a_1a_3a_2$, its codeword would be 010011110.

Table 3. Symbols showing their probabilities and the modified cumulative frequency along with codeword

Symbol	$p(x_i)$	$\lfloor \bar{F}(x_i) \rfloor_{l(xi)}$	Codeword
a_1	0.3	0.3	010
a_2	0.5	0.55	10
a_3	0.2	0.9	0111

1.4. Arithmetic Coding

Arithmetic coding is one of the well-known compression techniques, an effective mechanism for removing redundancy in the encoded data. Unlike other compression algorithms, it encodes the string of symbols, i.e., the message, instead of encoding each symbol in the symbol set. Arithmetic coding uses a range of values to uniquely represent any sequence of symbols. The range is proportional to the probabilities of the sequence of the symbols. So, even if the pmf of the source is unknown Arithmetic coding can produce a unique mechanism to code the messages given out by the source. [7].

Example of Arithmetic Coding

The range used in Arithmetic coding is (0, 1], where "(" denotes the closed interval and "]" represents the open interval. Therefore, any sequence is mapped to a range in the [0,1) line. Let a source alphabet be $X = \{a_1, a_2, a_3\}$. Now, suppose we have a source outputting symbols from the set, and each symbol output is independent from the others. Let the probability of occurrence of a_1 be $p(a_1) = 0.5$, that of a_2 be 0.3, and that of a_3 be 0.2. The range (0,1] is divided in the ratio of the probabilities as shown in Figure 1.2.

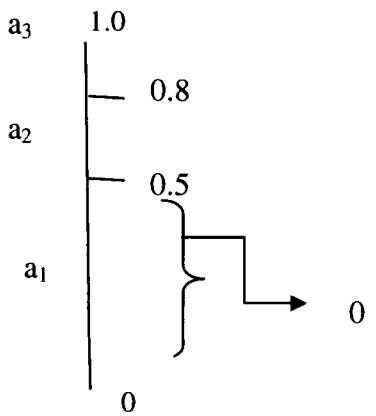

Figure 1.2. Symbols showing its distribution in (0,1] line in ratio of their probabilities.

The output bits given by the encoder follow the pattern shown in Figure 1.3. The figure shows that, if a selected range lies below 0.5 in the (0,1] line, the encoder outputs 0, whereas if it is above 0.5, it outputs 1. Similarly, the range (0.5,1] and (0,0.5] can be divided and subdivided, and the process can continue till the entire message is encoded by the encoder. Figure 1.3 illustrates such division and subdivision as well as the corresponding output bits[7].

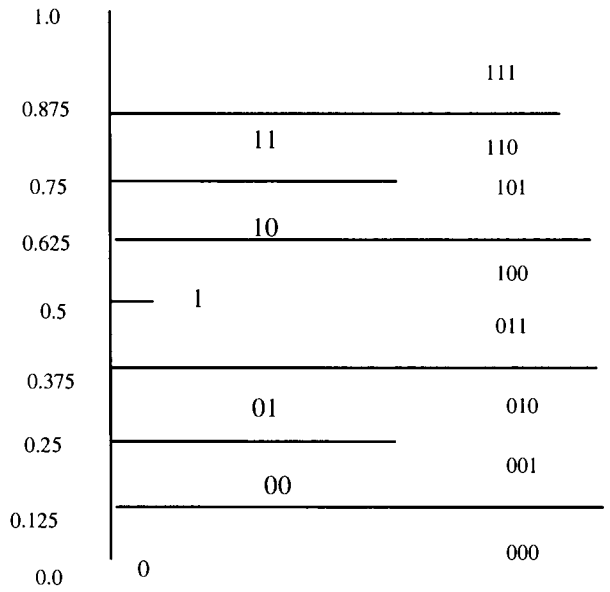

Figure 1.3. Representation of model showing the output bits.

Let us consider the above example where we have a symbol set containing three symbols: a_1, a_2, and a_3. From Figure 1.2, we know that each range uniquely represents the respective symbol. Therefore, the (0,1] line is divided in the ratio of the probabilities of the symbols, i.e., 5:3:2. Symbol a_1 is represented by the range [0,0.5). Similarly, a_2 is represented as [0.5,0.8) and a_3 as [0.8,1). Now, if a message, $\boldsymbol{a_1a_3a_2}$, needs to be encoded, the encoder would scan the first symbol which is a_1 in this case. The range in which the message would fall is [0,0.5). Referring to Figure 1.3, we know the encoder would output a

bit 0. The selected range is again subdivided into the ratio of the probabilities of the source symbols. Figure 1.4 shows the changed model with the new ranges in it. Because the next symbol in the message is a_3, the selected range becomes (0.4,0.5] as shown in Figure 1.4. The new range lies above 0.25; therefore, the encoder would output a bit 1; it is also above 0.375; therefore, another bit 1 is given out by the encoder. The new interval [0.4,0.5) is again divided, and the final range becomes [0.45,0.47) (as shown in Figure 1.5). The corresponding code becomes 0111. Therefore, the codeword for the message becomes 0111.

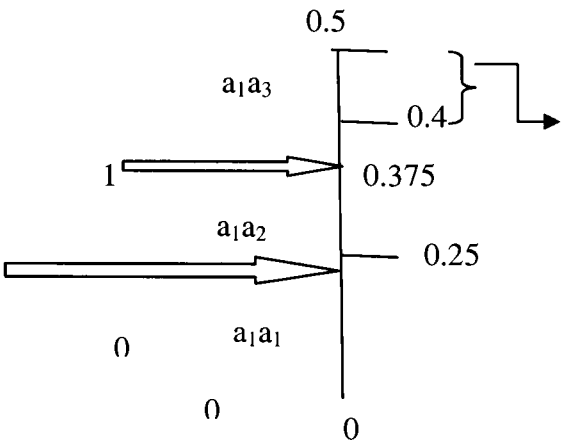

Figure 1.4. Arrangement of symbols after the second symbol is given out by the source.

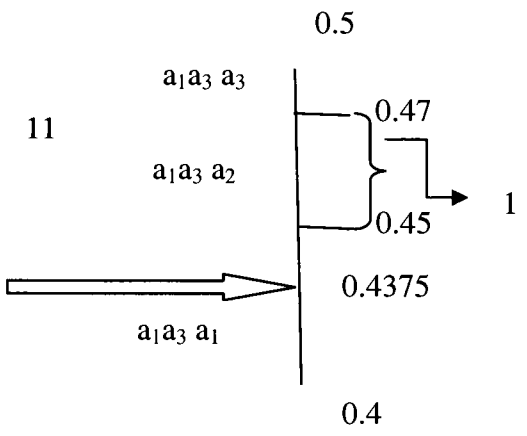

Figure 1.5. Arrangement of symbols after the third symbol is given out by the source.

There are basically two models of Arithmetic coding: Fixed model and Adaptive model. The above example shows the fixed-model Arithmetic coding technique. The fixed model is applied for cases where the source pmf is known to the encoder and decoder. On the other hand, the adaptive model assumes the initial distribution to be uniform. After encoding each symbol, the adaptive encoder updates its distribution. More frequent

symbols have their probabilities increased while the others have their probabilities decreased.

Let us perform Adaptive Arithmetic coding on the above example. We have a symbol set, $X = \{a_1, a_2, a_3\}$, where we do not know the distribution of the source, so we start with a uniform distribution. This implies that each symbol would have a probability of 0.33, i.e., $p(a_1)=p(a_2)=p(a_3)=0.33$. After encoding each symbol, the probabilities would be updated using the Laplace formula stated in equation (4).

$$\frac{(F_i+1)}{\sum_{i=1}^{n}(F_i+1)} \quad (1.4)$$

where F_i is the number of occurrence of symbol i and n is the number of symbols in the symbol set.

Let us assume the same message, $\boldsymbol{a_1a_3a_2}$, is given out by the source. If we redraw Figure 1.2, it would be as follows (Figure 1.6). Therefore, the output bit would be 0 as the selected range lies below the 0.5 line.

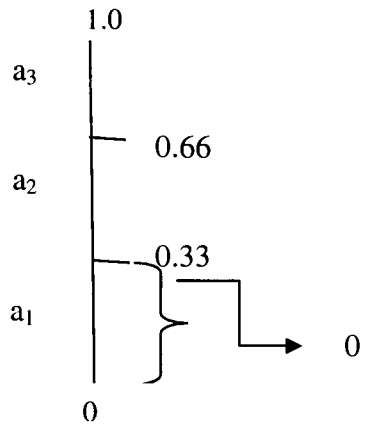

Fig 1.6 Arrangement of the symbols in ratio of their probabilities in (0,1] line.

After the encoding, the probabilities of each symbol are updated using the formula in equation (4). So far, the number of occurrences of $\boldsymbol{a_1}$ is 1 in the message, which means

F_{a1}=1, and so far, the number of symbols that occurred is also 1. Putting it into equation (4), we get $p(\boldsymbol{a_1})= \frac{1+1}{1+3}$ =0.5. Similarly, for the symbol $\boldsymbol{a_2}$, the probability becomes $p(\boldsymbol{a_2})= \frac{0+1}{1+3}$ = 0.25, and that of $\boldsymbol{a_3}$ is 0.25, too. Therefore, it can be observed that, since $\boldsymbol{a_1}$ has already occurred in the message, its probability is increasing while that of other symbols is decreasing. The new ratio of probabilities for the symbols becomes 2:1:1, and the selected range from Figure 1.6 would be divided into this ratio as in Figure 1.7.

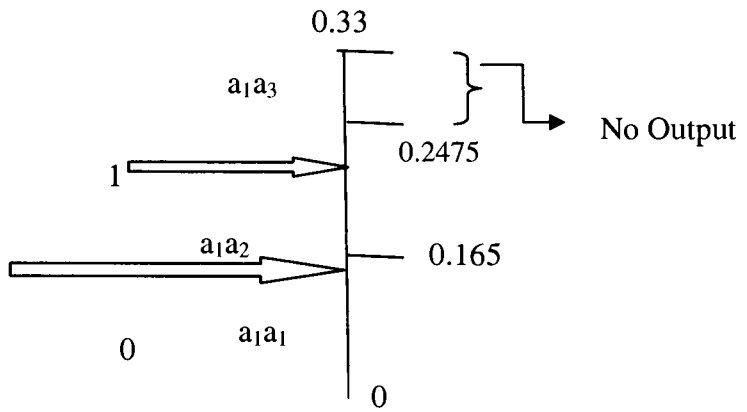

Figure 1.7. Arrangement of symbols after the second symbol is given out by the source.

The selected range (0.2475,0.33] does not lie completely under 0.25 or above 0.25, so the encoder is not able to decide on the bit; therefore, it does not output any bit. The probabilities of the symbols are updated, and the new probabilities become $p(\boldsymbol{a_1})$= 0.4, $p(\boldsymbol{a_2})$= 0.2, and $p(\boldsymbol{a_3})$= 0.4. The selected range is subdivided in the new ratio of probabilities. The encoder reads the next symbol given in the message, and because it is a_2, the selected range becomes (0.2805,0.3135]. The selected range lies above 0.25, so 1 is output by encoder, and also, it lies below 0.375, so it gives out 0 as shown in figure 1.8.Therefore, the code becomes 010, which is different than Normal Arithmetic coding.

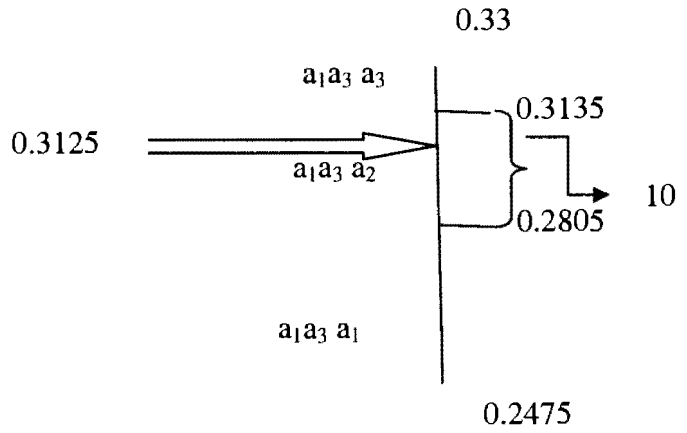

Figure 1.8. Showing the final range of the codeword for the message $a_1a_3\ a_2$

Here, we discuss a few advantages and disadvantages of Arithmetic coding. A few salient features of Arithmetic coding are as follows.

i) Uniqueness of representation: The codeword produced by Arithmetic coding is actually unique as, in Arithmetic coding, the entire message is encoded instead of finding the codeword for each symbol.

ii) Optimum compression: Arithmetic coding produces optimum compression compared to other compression techniques.

iii) Dynamic sources: Arithmetic coding can deal with dynamic sources with ease, i.e., the cases where the distribution of sources is not known. Adaptive Arithmetic coding does not need the source distribution. It updates its model during the encoding process, depending on the nature of the message.

iv) Separation of coding and source modeling: In Arithmetic coding, there is a clear separation between the source modeling and coding. We have already discussed that there can be two types of model in arithmetic coding, fixed model and adaptive model. A user can choose the model according to his

needs. The coding principle remains the same whatever be the source model. This facilitates dealing with complex models at ease.

Drawbacks of Arithmetic Coding

There are drawbacks for Arithmetic coding:

i) One of the main drawbacks in Arithmetic coding is its slow operation. There are too many calculations taking place while encoding or decoding a process, thereby increasing the processing time.

ii) One more problem of Arithmetic coding is that it is sensitive to error. A few bits of error may result in a completely different sequence. There are a few error-correcting techniques introduced to overcome this problem.

CHAPTER 2. JOINT COMPRESSION AND ENCRYPTION

In this era of information technology, exchange of information plays an important role in different businesses like online banking, online trade, and multimedia. These data are generally confidential and should be protected from eavesdroppers. Besides, the data need to be reduced in size to minimize the cost of transmission in terms of speed and time. Therefore, joint encryption and compression have gained much importance. Moreover, compressions also help to decrease the redundancy in the plaintext which makes the data more resistant to statistical method of cryptanalysis. Joint compression and encryption have also acquired increasingly important roles in medical-, commercial-, and government-related computing. Other applications where encryption is used for authentication and copyright protections are digital watermarking and wireless transaction.

2.1. Review of Present Techniques

In need of higher security and an exchange of a large amount of data, a lot of research work is going on to develop algorithms which could perform compression and encryption together. A few of them end up compromising the compression ratio while the other schemes turn out to be complex and increase transmission delay. A brief discussion of this research work is presented in this section.

In 1994, Finnila [8] showed an innovative technique to implement joint compression and encryption. In this paper, a fixed-library Autosophy tree network data compression has been combined with encryption using the library as the code key. As stated in the paper, the word "Autosophy" is formed from the Greek words "autos" (self) and "Sophia" (knowledge or wisdom) and can be translated as self learning or self knowledge. Here, algorithms are developed for self-assembling data networks in electronic

memories where all new information is learned from the little informations known.Each serial tree network starts with a seed word which could identify the serial tree network. Each memory word consists of two parts; the gate and the pointers. The gate consists of the data while the pointers contains the address of the previous node. During encryption, the serial network matches the characters in words and supplies the identified word or part of the word. The method here provides a fixed library compression which includes encryption, too. The applications of the method include the public telephone system and news reporting for a national weekly magazine which does not want its competitor to see its future stories. After reviewing this particular paper, it has been observed that, for better data security, frequent changes of a library need to be done; these changes may be difficult and not cost effective for maintenance.

Pseudo random numbers can be effectively used in cryptography. In [9], we came across a few approaches where pseudo random shuffle is used in Huffman coding or Arithmetic coding. In the case of Arithmetic coding, pseudo random shuffle is used to shuffle the interval table for Arithmetic coding before performing the compression. The idea behind Lempel-Ziv compression is to replace a group of consecutive characters with an index in the dictionary [10]. The algorithm [9] first initializes a pseudo random number generator with the encryption key. It shuffles the initial values and performs Lempel-Ziv compression. In spite of using a pseudo random shuffle and instilling some randomness in the method, it is not a very strong encryption technique.

There are other efficient multimedia encryption techniques using entropy codec design as discussed in paper [11]. The method described in [11] needs to generate and store a very huge number of statistical tables to achieve security. A codeword is selected

randomly from these large numbers of statistical tables. Such a statistical table grows with the number of symbols, making it unsuitable for compressing and encrypting a very large symbol set.

In April 2006, a new method of compression and encryption scheme using variable model Arithmetic coding and a coupled chaotic system was presented. The work incorporated recent results of the chaos theory, which were proved to be cryptographically secure. An idea has been proposed which develops an encryption technique with the use of a Couple Chaotic System (CCS) Pseudorandom Bit Generator (PRBG) for the generation of a bit string. CCS is used to generate random numbers, and the statistical model of Arithmetic coding is changed according to the random number. Chaotic maps are utilized to initialize the PRBG. Every time the model tries to encode a symbol, it permutes the distribution in a complex manner which introduces delay and uses memory space. Though the scheme produces a compression between 67.5% and 70.5%, it is not viable for large messages due to the delay and large computation cost. Moreover, the chaotic system is quite complex with a high implementation cost that may restrict wide usage[12]. The security claim of the system does not provide any mathematical proof. It is purely experimental.

There have been other approaches of attaining joint compression and encryption using Arithmetic coding. One of them uses splitting of the intervals with permutations at both input and output [13][14]. The paper proposes a coding technique which has very strong encryption capabilities at the expense of efficiency. Security of Arithmetic coding is widely studied and attacks have also been proposed to comment on its cryptographic capabilities [15].

Joint compression and encryption is widely used in JPEG 2000. In [16] and [17] we come across efficient algorithms of compressing JPEG 2000 without compromising compression ratio. Research shows that one of the areas where joint compression and encryption is applied widely is Multimedia and video encryption [18]. In these types of encryptions, compression algorithms like Huffman coding and Arithmetic coding are used. Recent research also shows cryptanalysis of techniques used in MPEG video encryption and methods to improve its encryption mechanisms. Encryption with Multimedia compression at a low computational cost has been proposed in paper [19][20][21]. Huffman coding and QM coder are used to build such encryption schemes which also prove to have good compression capabilities.

The approaches used for encryption of textual data may not be suitable for multimedia data because, in most natural images, the values of neighboring pixels are strongly related. A new approach has been presented in a recent research where a new permutation technique is introduced based on a combination of image permutation and encryption algorithm named after RijnDael. Dividing the original image into a 4 pixel X 4 pixel block, then applying permutation in these blocks, and encrypting using the RijnDael algorithm, plain images can be encrypted. Joint compression and encryption is also applied to sensor networks where AES is used as one the encryption schemes [22].

One of the well-known compression techniques, Shannon Fano Elias coding, can be used as a good encryption technique in different applications like multimedia and communication systems. Such a technique is observed in [5] where SFE is utilized for an encryption algorithm using the order of the source symbols as the key. As discussed in

Section 1.3, we know that SFE codewords depend on the order of the source symbols in the symbol set. Different orders produce different codeword sets. Therefore, before transmission, a particular order is picked up and exchanged between the sender and the receiver. Encoding and decoding is done using the same order. Cryptanalysis of the algorithm presented in the same paper suggests that, for an attacker to find the right order (i.e., the key), he needs to have all the candidate codeword sets in his hand. By candidate codeword set, we mean codeword sets for all the orders of the symbol possible. For example, for a codeword set of n symbols, the number of candidate codeword sets would be $n!$. The attacker needs to scan the encoded string from left to right and then apply the process of enumeration to find the right order. For a large value of n, it is quite difficult for an attacker to find $n!$ codeword sets. On the other hand, for a small symbol set, an attacker can easily find the right key. Here lies the disadvantage of this method. The work which is done in the thesis has overcome the disadvantage of the mentioned algorithm. The technique which the cryptanalyst of [5] follows cannot break the algorithm described in Chapter 3. The reasons are discussed in the next chapter.

CHAPTER 3. SECURITY USING SHANNON FANO ELIAS CODING

3.1. Introduction to SFE Encoding

The use of Shannon Fano Elias (SFE) code for encryption can be applied in multimedia and communication systems. The advantage of an SFE codeword is that it depends on the order in which the symbols are arranged in the codeword set and does not matter if the probability mass functions of the symbols are known to everyone. Such an advantage proves to be fruitful in cases like compression of English text where the pmf) of the source symbols are known. SFE codes rely on ordering the source symbols for obtaining codewords. Different orders result in different codeword sets, thus enabling the usage of ordering as the key for encryption. Let us assume that a source is putting out a string of symbols, $S = s_1, s_2,\ldots,s_m$, where $s_i \in \mathcal{X} = \{x_1, x_2,\ldots, x_n\}$. Set $\mathcal{X}$ is called the source alphabet. The pmf of $\mathcal{X}$ is $p(x_i) > 0$. An ordering of alphabet $\mathcal{X}$ is a permutation of the symbols $\{x_1, x_2,\ldots,x_n\}$. The construction of SFE is based on finding the modified cumulative frequency of all the symbols in the symbol set. The modified cumulative distribution function is defined as

$$\bar{F}(x_i) = \sum_{k=1}^{i-1} p(x_k) + \frac{1}{2} p(x_i) . \qquad (3.1)$$

$\bar{F}(.)$ represents the sum of probabilities of all symbols placed before x_i in the ordering plus half of the probability of x_i. Because the random variable is discrete, the cumulative function consists of step size of $p(x_i)$. Therefore, we can determine x_i from $\bar{F}(x_i)$. In general, $\bar{F}(x_i)$ is a real number only expressible by an infinite number of bits. Therefore,

$\bar{F}(x_i)$ is rounded to $l(x_i) = \left\lceil \log \frac{1}{p(x_i)} \right\rceil + 1$ bits, denoted by $[\bar{F}(x_i)]_{l(xi)}$. SFE coding uses $[\bar{F}(x_i)]_{l(xi)}$ as the codeword for x_i. The resulting codewords are proven to be prefix-free [4]. The expected length, L, of an SFE-encoded sequence is defined as

$$L = \sum_{i=1}^{n} p(x_i) l(x_i) \,. \tag{3.2}$$

It can be shown that L satisfies $H+1<=L<H+2$, where H represents entropy of the source [4]. In [5] redundant bits were removed from SFE codewords to make them shorter and, hence, reduce L.

It is implied from the above discussion that, because $\bar{F}(x_i)$ depends on the position of the symbol x_i in a particular ordering, different orderings would lead to different values of $\bar{F}(x_i)$ and, hence, different codeword for each x_i. For a symbol set of n, there could be $n!$ possible orderings. Therefore, if an ordering is picked as a key, then SFE could be used for encryption [5]. Cryptanalysis of such an approach has shown that SFE encoding has time complexity, $O\left(n! \times \frac{1}{\prod_{i=1}^{n} p(xi)}\right)$. This turns out to be small if the alphabet size, n, is small. For a small n, it is possible to find all $n!$ codeword sets. A technique has also been proposed in [5] which could break the SFE code if codeword sets are available to the attacker.

3.2. Adaptive SFE Code

The thesis proposes a new approach which makes it difficult for an attacker even when there is a small n. The new method is called Adaptive SFE as it changes the pmf each time t symbols are output by the source. Initially, the pmf is assumed to be uniform, that is, $p(x_i)=1/n$. The SFE codewords obtained with this distribution are used to encode symbols s_1 through s_t. The pmf $p(x_i)$ is updated according to the Laplace rule [7] as follows. Let the number of times x_i occurs in the sequence received so far be F_i. The

updated value of $p(x_i)$ is $\frac{(F_i+1)}{\sum_{i=1}^{n}(F_i+1)}$. These updated values are then used to encode the next t symbols output by the source. The pmf is, therefore, updated m/t times during compression of a file with m symbols. Adaptive SFE also uses two of the orderings (i.e., two permutations on $\{x_1, x_2,\ldots,x_n\}$) as the key, K_0 and K_1, while encoding each symbol, s_i. Therefore, the SFE code of symbol s_i depends on the ordering of K_{b_i}, where b_i is the i^{th} bit output by a secure pseudorandom bit sequence generator (PRBG). Cryptanalysis of this new adaptive SFE with two orderings has a time complexity of $O(2^t(n!)^2)$. It is also observed that t is proportional to m, the length of the string of symbols being compressed, and, therefore, is a very large number (t>500). Our experiments on English text sources show that the compression is the same for values of t between 1 and $m/10$. Since $m>>10{,}000$, we conclude that our new method is superior in terms of security (time complexity of $O(2^{1000}(n!)^2)$) without sacrificing compression.

3.3. Construction of Adaptive SFE Code

The SFE encoder requires a pmf D over the alphabet and an ordering in order to find the codeword for symbol s_i output by the information source. The adaptive SFE algorithm follows the standard SFE encoding with some changes to it.

i) The algorithm, at first, selects two orderings, K_0 and K_1, at random from the set of all $n!$ permutations of $\{x_1,x_2, \ldots,x_n\}$. Based on the output of the pseudorandom bit sequence generator, one of these orderings is used to obtain SFE codewords for a given pmf D.

ii) The initial pmf D is the uniform distribution, $p(x_i)=1/n$. Every t symbols, the pmf D is updated according to Laplace's rule as follows: $p(x_i) = \frac{(F_i+1)}{\sum_{i=1}^{n}(F_i+1)}$,

where F_i is the number of occurrences of x_i in the sequence received so far. If the total number of symbols in the file being compressed is m, the pmf is updated m/t times. These changes lead to the following encoding algorithm (Algorithm 1).

Algorithm1:- Encoding for the new Adaptive SFE technique

Input: - File to be compressed containing symbols $S = s_1, s_2, \ldots, s_m$, where $s_i \in \mathcal{X} = \{x_1, x_2, \ldots, x_n\}$. PRBG seed, orderings K_0, K_1 chosen at random, and integer t.
Output:- The compressed binary file.

- $j = 1$, initial distribution D is uniform, $p(x_i) = 1/n$, $1 \leq i \leq n$

while j ≤ m **do**

- Generate next pseudo-random bit b using the PRBG.
- Read symbol s_j from the input file and find its position in the ordering K_b.
- Using K_b and current pmf D find the SFE codeword for s_j. Append this codeword to the output file.
- If $j = kt$ for some integer k then update pmf D according to Laplace's Rule.
- j = j + 1

end while

Encoding Example

The above algorithm is explained via the following example. A source is considered with alphabet set $\{x_1, x_2, x_3\} = \{A, B, C\}$. Initially, pmf D is $p(A)=p(B)=p(C)=0.33$*(approximately)*. The codewords, $C(x_i)$, using the SFE method for the six orderings are shown in Table 4. For this example, let us assume that the PRBG generates the following pseudorandom bits: 1100. Let the two orderings chosen at random be $K_0 = ACB$ and $K_1 = ABC$, and let t=1. (The pmf D is updated after each symbol is read from the file to be compressed.) Let the file to be compressed consist of the symbols $(s_1, s_2, s_3) = B, A, C$.

Table 4. Codewords for different orderings

Ordering	$C(A)$	$C(B)$	$C(C)$
ABC	00	01	11
BAC	01	00	11
ACB	00	11	01
BCA	11	00	01
CAB	01	11	00
CBA	11	01	00

Encoding proceeds as follows.

i) The first PRBG bit is 1; therefore, the ordering is $K_1 = ABC$. The initial pmf D is uniform. From Table 4, the codeword for $s_1 = B$ is $C(B)$ =01.

ii) The next PRBG bit is 1. Therefore, the key is $K_1 = ABC$. Because the first symbol received was B and t=1, we update distribution D to $p(A)$=0.25, $p(B)$=0.5, and $p(C)$=0.25. Using SFE, the new codewords are $C(A)$ = 00, $C(B)$ = 10, and $C(C)$ =10. Therefore, the codeword for $s_2 = A$ is 00.

iii) The next PRBG bit is 0; therefore, the key is $K_0 = ACB$. Because the first two symbols received were BA and t =1, we update distribution D to $p(A)$ = 0.4, $p(B)$ =0.4 and $p(C)$ = 0.2. Using SFE, the new codewords are C(A) = 00, C(B) = 11, and C(C) = 10. Therefore, the codeword for $s3$ = C is 10.

The output of the encoder is, therefore, $C(BAC) = 010010$. Decoding can be performed by reversing the encoding process and is given in Algorithm 2. The decoding algorithm also has two changes as mentioned in the encoding algorithm.

Algorithm 2: Decoding for the new Adaptive SFE technique

Input: Compressed binary file with bits $(B_1, B_2,..., B_r)$. PRBG seed, orderings K_0 , K_1,
and integer t.

Output: The uncompressed file with symbols $S = s_1, s_2,..., s_m$.

- $j = 1$, initial distribution D is uniform, $p(x_i) = 1/n$, $1 \leq i \leq n$
- The current bit being scanned is B_1.

while $j \leq r$ **do**

- Generate next pseudo-random bit b using the PRBG.
- Scan the compressed binary file from left to right. Assume the bits $(B_1, B_2,...., B_j)$ have already been decoded. Starting from the current bit being scanned, B_j, find the **minimum number of bits**, q, such that, $0.B_{j+1}B_{j+2}.....Bj+q \leq 0.B_{j+1}B_{j+2}.....Bj+q + 2^{-q}$ for **exactly one** x_i.
- Append this symbol x_i to the output file.
- If the number of symbols output so far is kt for some integer k then update pmf D according to Laplace's Rule.
- $j = j + q$

end while

Decoding Example

Following Algorithm 2, decoding of the first symbol is described below for when the received sequence is 010010 and the pseudo random bit sequence is 1100.

i) The initial pmf is $p(A) = p(B) = p(C) = 0.33(approximately)$. Because the first pseudo-random bit is 1, the ordering is $K_1 = ABC$. The modified cumulative frequencies are $\bar{F}(A) = 0.167$, $\bar{F}(B) = 0.497$, and $\bar{F}(C) = 0.827$.

ii) After reading the first bit 0 in the received sequence, both symbols A and B satisfy $0.0_2 \leq \bar{F}(A \text{ or } B) \leq 0.1_2$. (A subscript of 2 indicates a binary

number.) Since there is no unique symbol that satisfies the inequality, the next bit is read.

iii) Now the sequence read so far is 01, and the following condition is satisfied by exactly one symbol, namely B, $0.01_2 \leq \bar{F}(B) < 0.1_2$. Therefore, the decoder outputs B.

iv) The new pmf is computed as ($p(A)$ =0.25, $p(B)$= 0.5 $p(C)$ =0.25), and the decoding steps are repeated.

The complexity of the encoding algorithm is obtained based on the number of times new probabilities are computed. When a new pmf is updated, n new probabilities are computed. In our algorithm, the probabilities are, therefore, computed nm/t times. The non-adaptive SFE method, on the other hand, computes the cumulative probabilities only n times in the beginning of the algorithm. However, the non-adaptive SFE method has the drawback of not being able to compress information sequences with unknown pmf.

3.4. Compression Capabilities of Adaptive SFE

The new adaptive SFE compression method has been applied on English text files of varying sizes. In each case, a non-adaptive SFE compression, an adaptive SFE compression with $t = 1$, and an adaptive SFE compression with $t = t_{max}$ has been performed. It has been noted that, when $t = 1$, the pmf D is updated every time a symbol is read from the text file. It has also been found that the maximum value of t that results in a compression percentage is the same as that when $t = 1$. We call this value $t=t_{max}$. Values of t greater than 1000 have not been tried. The results are given in Table 5.

Table 5. Compression percentage for various SFE methods

Input File Size	% Compression		
	Non-Adaptive	Adaptive (t=1)	Adaptive (t_{max})
968,789	44	42.2	42.2,(1000)
6,313	43.8	42	42,(650)
9,715	44	43	43,(500)
2,500	55	71	71,(100)

In Table 5, the compression percentage is calculated as follows:

$\frac{Original\ size\ in\ bits - Compresses\ size\ in\ bits}{Original\ size\ in\ bits}$ X 100. It has been noted that the original size of the file in bits is found by assuming that each alphabet in the English language requires 5 bits. In all cases, $t_{max}= m/10$, where m is the number of symbols in the text files being compressed. Compared to the non-adaptive SFE method, the adaptive SFE method has a loss of compression of about 1 to 2%. However, the adaptive method has the advantage of being able to compress sources with unknown pmfs. The bottom entry in Table 5 is a text file with 2500 English alphabets of unknown distribution. The proposed method outperforms the non-adaptive method by 16%. Table 5 shows that the value of t_{max} is large if the input file size is greater than 10,000 symbols. In the next section, we will see that the time complexity of a cryptanalysis on the adaptive SFE is proportional to $2^{t_{max}}$.

3.5. Cryptanalysis of the Adaptive SFE method

The cryptanalysis algorithm has as input a binary file that is the output of our adaptive SFE encoding algorithm (Algorithm 1). The cryptanalysis has to proceed by examining this file

from left to right. After a guess on the pair of orderings has been made, a symbol can be obtained using the decoding algorithm. Because the initial probability is uniform, the codeword sets are permutations of the same codewords for any ordering (for an example refer to Table 4). Thus, as long as the pmf is uniform and the PRBG unknown, in each step, the attacker has to perform decoding for both orderings K_0 and K_1. The attacker, therefore, computes 2^t possible decodings for the first t symbols she outputs. At this point, she changes the pmf, D for each one of these 2^t decodings and continues the process. If the new pmf is skewed (away from uniform), then the chances that the bits being read from the input file do not match any codeword (This happens if the wrong ordering is chosen.) are high. If this happens, we eliminate such an ordering and start all over again with a new choice for the pair of orderings. The number of choices for the pair of orderings is $\frac{n!(n!-1)}{2}$, where n is the size of the alphabet. This cryptanalysis is described in Algorithm 3.

Algorithm 3: An attack for our new Adaptive SFE technique

Input: Compressed binary file with bits $(B_1,B_2,\ldots\ldots, B_r)$. Integer t.
Output: The uncompressed file with symbols $S = s_1, s_2,\ldots\ldots, s_m$. Let set G contain every possible decoding of the input file observed so far. Set G = ∅.
Initial pmf D is uniform, $p(x_i) = 1/n$; $1 \leq i \leq n$.
1) Choose a new pair of orderings (K_0,K_1). For every $d \in G$, obtain pmf D (using Laplace's rule). Obtain SFE codeword tables for K_0 and K_1 using pmf D.
2) For every $d \in G$, do the following. Do decoding until the next t symbols are output. Let d1,d2,.....,dk, (k ≤ t) be decoded starting with d. G = (G/{d}) ∪{d|d1, d|d2,....d|dk}, where d|di denotes the concatenation of d and di. If there exists a d with k = 0, then remove it from set A.
3) If the number of decodings (k) for every $d \in G$ is 0, then the current choice of (K_0, K_1) is invalid. Go to step 1.
4) If $|G| > 1$ and the input file has not been completely read, go to step 2.
5) If $|G| > 1$ and the input file has been completely read, then the attack is complete, and G contains all possible decodings of the input file.
6) If $|G| = 1$ and the entire input file has been decoded, we have success, and set G contains the correct decoding.

Algorithm 3 is illustrated by mounting an attack on the example of the previous section. Suppose the attacker received the bit sequence 010010. Because there are only three symbols, the initial uniform distribution, D, is $p(A) = p(B) = p(C) = 1/3=0.33(approx)$. The codeword sets for every possible ordering of A, B, and C was found in Table 4. The pseudorandom sequence used in the encoding is unknown to the attacker. Initially, G =$\emptyset$. It is assumed that the attacker chooses orderings $(K_0, K_1) = (ABC, BAC)$. The pmf D = $(p(A), p(B),p(C)) = (0.33, 0.33, 0.33)$. Let $t = 1$. Using ABC as the order of the symbols, 01 is decoded as B, and using BAC ordering, 01 is decoded as A. Thus, G = $\{B,A\}$. If B$\in$ G is considered, then the next two bits, 00, are decoded as A assuming the order as ABC but if we assume the order to be BAC then the decoding would not have been possible . Thus, set G becomes G = $\{BA,A\}$. Similarly if $A \in G$ is considered, then 00 cannot be decoded assuming the order as ABC but if we assume the order to be BAC then 00 is decoded as B. Thus, G = $\{BA,AB\}$. After decoding the next two bits, 10, set G becomes, G = $\{BAB, BAA, ABB, ABA\}$. This process is shown in Table 6. It is noted that the SFE codewords used correspond to the enhanced or shortened codewords from [9,10]. At each step of the attack, both orderings K_0 and K_1 have to be tried. This has to be repeated for all $\frac{n!(n!-1)}{2}$ possible pairs of orderings. Thus, the complexity of the attack is $2^{m\frac{n!(n!-1)}{2}}$, where m is the size of the file being compressed and n is the number of symbols in the alphabet set. From Table 6, it is noticed that, for some orderings, the decoding ends earlier than others. It is guaranteed that all decodings are alive for the first t steps because the initial pmf is uniform. Thus, the minimum complexity of the attack is $2^{t\frac{n!(n!-1)}{2}}$. If we assume that $t \geq m/10$, then the minimum

complexity is $2^{m/10}\frac{n!(n!-1)}{2}$. Thus, it is impossible to mount an attack on the adaptive SFE method. Even if the attacker were successful, she only finds K_0 and K_1, and still will not be able to decode any other file because she does not know the pseudorandom bit sequence.

Table 6. Attack on string 010010

Bits Read	Key	Output	Bits Read	Key	Output	Bits Read	Key	Output
01	ABC	B	00	ABC	A	10	ABC	B
							BAC	A
			00	BAC	--		ABC	--
							BAC	--
01	BAC	A	00	ABC	--		ABC	--
							BAC	--
			00	BAC	B	10	ABC	B
							BAC	A

One of the main reasons for obtaining this improvement is the fact that the adaptive method has forced the attacker to decode from left to right, whereas previous non-adaptive methods could look for a codeword in the entire file being decoded. Note that the PRBG bits are only used to select one of the orderings, K_0 or K_1. Thus, it is hard for the attacker to find the PRBG bits even if many plaintext-ciphertext pairs are known. This enables the use of weaker pseudo-random sequence generators.

3.6. Conclusion

Thus, it can be concluded that the new adaptive SFE compression algorithm can also be used for encryption. The new algorithm uses the following two new concepts to improve security:

i) It uses two orderings, K_0 and K_1, one of which is chosen for the compression based on the output of a pseudorandom bit sequence generator.

ii) The probability mass function, D, used in the encoding is updated every t steps according to the frequency of symbols observed so far.

iii) The compression properties of the new algorithm are virtually the same as the simple SFE compression algorithm. However, the complexity of the attack increases dramatically to $2^{m/10}\frac{n!(n!-1)}{2}$, where m is the size of the file being compressed and n is the number of symbols in the alphabet set. The new adaptive SFE method requires probabilities *to be computed nm/t* times during a compression. This minor increase in computation allows the method to compress sources with unknown probability mass functions.

CHAPTER 4. SECURITY USING MODIFIED ARITHMETIC CODING

4.1 Introduction

Arithmetic coding is one of the strongest compression techniques used in various applications like image compression of jpeg files. We learned from Section 1.4 that Arithmetic coding is the binary representation of a range of real numbers between 0 and 1. This range is determined by dividing and subdividing the [0,1) line with respect to the probabilities of the symbols in the symbol set. Section 1.4 talked about two models of Arithmetic coding, fixed and adaptive models. The fixed model of Arithmetic coding utilizes the source distribution for encoding and decoding process. On the other hand, the Adaptive model proceeds by assuming the initial distribution to be uniform. It updates the distribution after encoding each symbol in the message. The probabilities of the symbols are updated based on the Laplace rule which states the updated probability of the symbol to be $\frac{(F_i+1)}{\sum_{i=1}^{n}(F_i+1)}$, where F_i is the number of occurrences of symbol i and n is the number of symbols in the symbol set. The idea behind it is that the probability of the symbol increases if it occurs more frequently in the message while those which occur less have less probability.

4.2. Construction of Modified Arithmetic Coding

This research work has utilized Adaptive Arithmetic coding for encryption. Adaptive Arithmetic coding has been modified, and a new algorithm has been proposed, hence the name is Modified Arithmetic Coding. In this section, we discuss the construction of Modified Arithmetic Coding, taking into consideration a symbol set containing two

symbols. The same algorithm can be extended to a larger symbol set with a slight modification. Modified Arithmetic Coding follows the standard Adaptive Arithmetic coding as discussed in Section 1.4 with the following changes in it:

i) Our symbol set has two symbols; therefore, there are two possible ways by which we can divide the range [0,1) as shown in Figure 4.1. Modified Arithmetic Coding uses the seed of a secure pseudorandom bit generator (PRBG) for its encryption.

ii) The seed is used to generate a pseudo random bit, and based on this bit, the order of the symbol is determined. For example, if the pseudo random bit is 0, then the order would be as shown in Figure 4.1a, and if the bit is 1, then the order is shown in Figure 4.1b.

(a) (b)

Figure 4.1. Two possible orders of the symbols.

iii) A constant, k, is also defined and determines the number of bits that need to be output before the order of the symbols is changed. For example, if k is 5, then the encoder would encode the message with a particular order and continue doing so till 5 bits are given out as output by the encoder.

iv) Both k and the seed of PRBG act as the key for this encryption method.

The above changes lead to the following encoding algorithm.

Algorithm 4. Encoding for the Modified Arithmetic Coding

Input: File to be compressed containing symbols $S = s_1, s_2, \ldots, s_m$ where $s_i \in X = \{0,1\}$. PRBG seed, and a constant k
Output: The compressed binary file.

- $j = 1$, initial distribution D is uniform, $p(x_i) = 1/n$, $1 \leq i \leq n$. Here, $n=2$.
- Generate the next pseudorandom bit b using the PRBG.

while j $\leq m$ **do**

- Read symbol s_j from the input file, and find its position in the ordering K_b.
- Using K_b output, the bit(s) corresponding to [0,1] line like Arithmetic coding.
- Update the probabilities using the Laplace formula. Append this codeword to the output file.
- If $j == k$,
  generate the next pseudo-random bit b using the PRBG.
- j = j + 1

end while

Encoding Example

Let us consider a source with a symbol set [0 , 1]. The source gives a string of binary bits which is encoded using Modified Adaptive Arithmetic Coding. The bits given by the source are assumed to be 1 1 0 1 0 0 in this example. Let us also assume that the first three bits of PRBG are 101. The two possible orders in which the symbols can be arranged are as follows in figure 4.2(a) and 4.2(b):

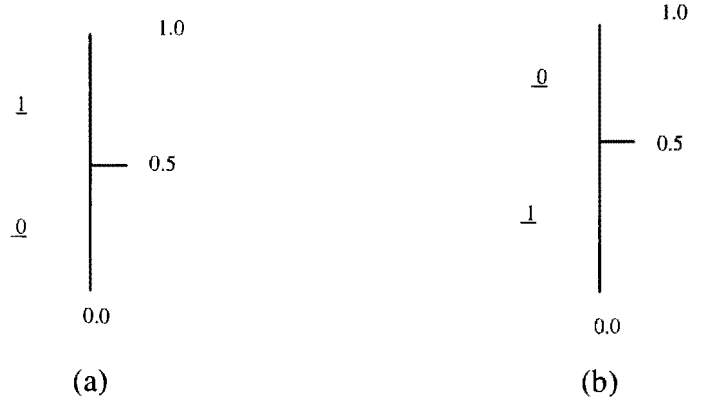

Fig4.2. Arrangement of the symbols in (0,1] line.

We take into assumption that, if a pseudo random bit is 1, the input symbols should be arranged according to order 4.2(a), whereas when a pseudo random bit is 0, they should be arranged according to order 4.2(b). Figure 4.3 shows the representation of output bits in Arithmetic model.

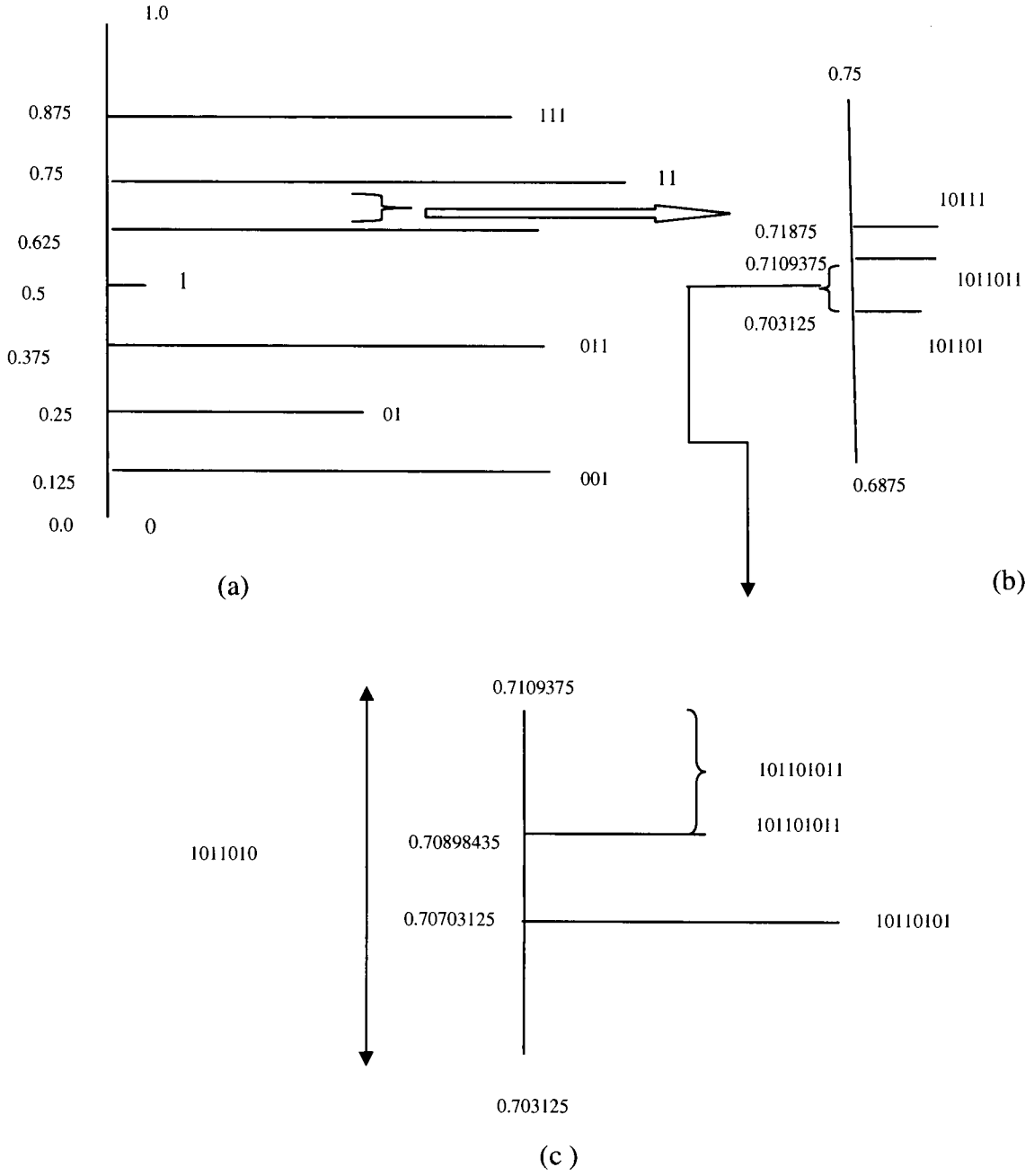

Fig 4.3 Representation of the output bits in the model.

Encoding proceeds as follows:

i) The initial distribution is uniform, i.e., $p(\underline{0})=p(\underline{1})=\frac{1}{2}=0.5$. The first PRBG bit is 1, so the order should be 4.2(a). The orders of the symbol arranged in a (0,1] line in the ratio of their probabilities would be as follows in fig 4.4.

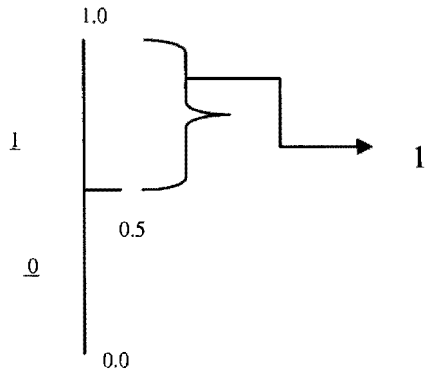

Fig 4.4.Representation of the symbols in the range (0,1] in ratio of their probabilities.

Since the first bit given by the source is 1, the selected range would be (0.5,1], and because the range lies completely above 0.5, the output binary bit should be 1 (referring to Figure 1.3 of Chapter 1).

ii) We update the probabilities following the Laplace formula, and the new probabilities are $p(\underline{0})= 0.33$, $p(\underline{1}) = 0.66$. The selected range (0.5,1] is divided into the ratio of the probabilities of the symbols as shown in Figure 4.5.

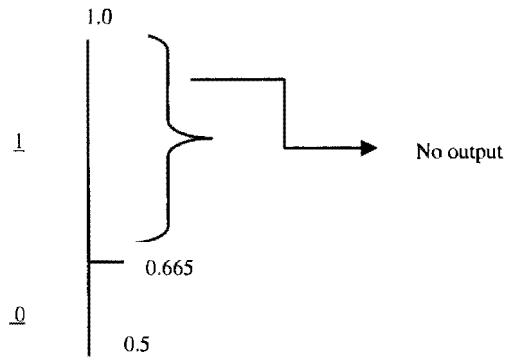

Fig 4.5.Representation of the symbols in the range (0.5,1] in ratio of the probabilities.

iii) The next bit given by the source is 1, so the selected range becomes (0.665,1], and because this range does not completely lie above 0.75 or below 0.75, the encoder is not able to decide the output bit and proceeds to read the next bit given by the source.

iv) The updated probabilities in this step become $p(\underline{0})$= 0.25, $p(\underline{1})$ = 0.75, which leads to the following model (Figure 4.6).

v) Since the next bit in the message is 0, the selected range becomes (0.665, 0.74875], and this range lies completely below 0.75; therefore, the encoder outputs a bit 0, and also because it lies completely above 0.625, it outputs a bit 1 simultaneously.

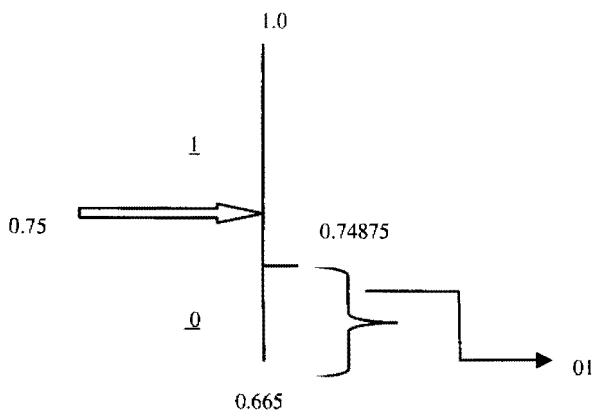

Figure 4.6. Representation of the symbols in the range (0.665,1] in ratio of their probabilities.

In this example, we assumed that k=3 which means that, after encoding every three symbols, the encoder would check the next bit of PRBG output and switch the order accordingly. As the first bit of PRBG was 1, the order followed so far was as per Figure 4.2(a). At this point, the encoder checks the next output of PRBG and finds that to be 0, which means the order should be switched and be as per Figure 4.2 (b). (i.e., the symbol 1 should be below and symbol 0 should be above.) From step 4), we know that the selected range was (0.665,0.74875], and the updated probabilities become $p(0)$= 0.4, $p(1)$ = 0.6. Figure 4.7 shows the division of the range in the ratio of their probabilities and also the new order.

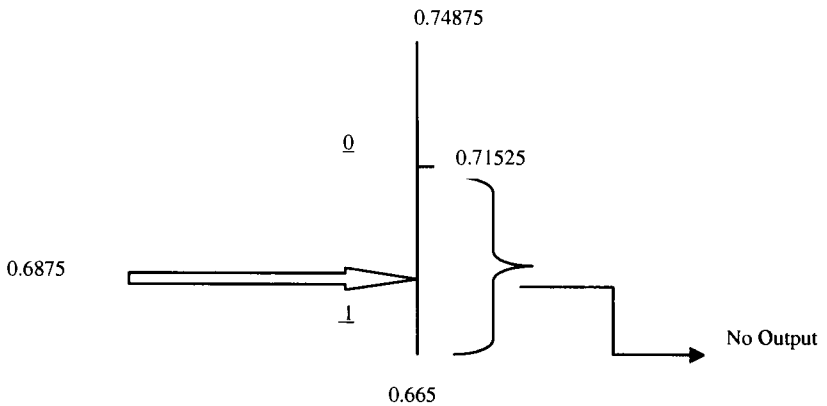

Figure 4.7. Representation of the symbols in the range (0.665,0.74875] in ratio of their probabilities.

The selected range is (0.665, 0.71525] as the next bit in the message is 1. The encoder is not able to output any bit because the range does not completely lie above or below 0.6875. Following the same rule, the probabilities are updated, and the selected range is divided. Figures 4.8 and 4.9 show the next two steps of encoding the next two symbols

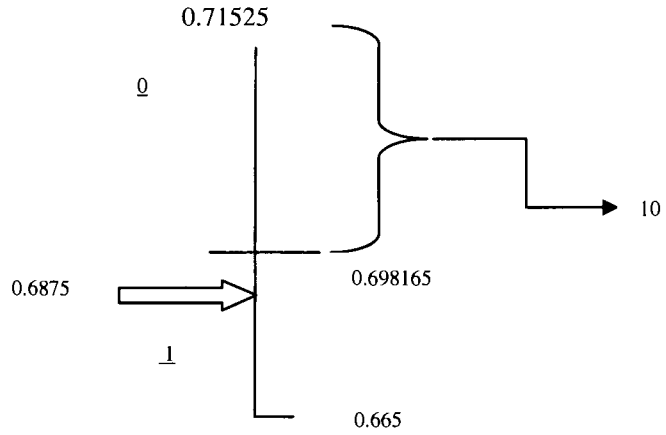

Fig 4.8. Representation of the symbols in the range (0.665,0.71525] in ratio of their probabilities.

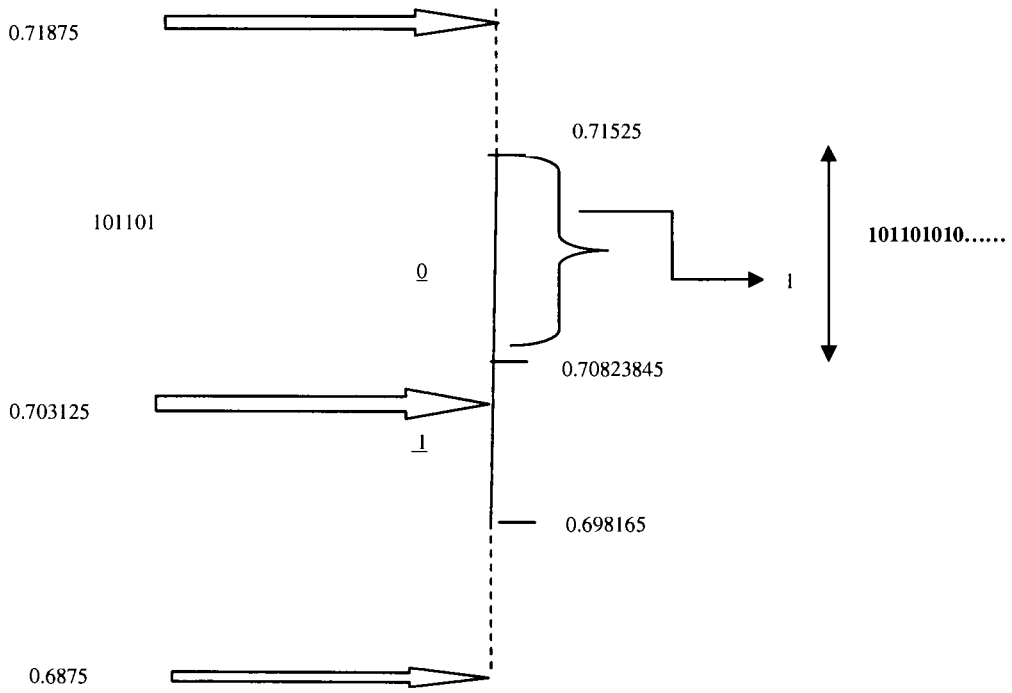

Fig 4.9. Representation of the symbols in the range (0.698165,0.71525] in ratio of their probabilities.

vi) The final range is (0.70823845, 0.71525], but this does not conclude our encoding process. Here comes an interesting part of encoding. The binary bits given by the encoder so far are 101101, but this string of binary bits does not uniquely represent the final interval we have obtained. Referring to Figure 4.3(c) we can see 101101 represents the range (0.703125, 0.71875]. Our final range lies completely within this above range, but we need to find a unique binary representation of the final range or its subset; that would be our codeword. Since the number 0.70823845 does not convert to a binary numbers with terminating bits, it is not possible to represent the range (0.70823845, 0.71525] with a binary string that has terminating bits. Therefore, as discussed in Chapter 1, we would find the biggest range that completely lies within the final range and can be represented by a binary string of terminating bits. From Figure 4.3(c), we see that the range (0.708984375,0.7109375] lies completely within the final range (0.70823845, 0.71525]. The binary representation of the range (0.708984375,0.7109375] is 101101011, and this is our final codeword for the message 1 1 0 1 0 0 sent to the encoder by the source. The decoding algorithm (Algorithm 5) for Modified Arithmetic coding is as follows.

Algorithm 5: Decoding for the Modified Arithmetic Coding

Input: File to be decoded containing symbols $C = c_1, c_2, \ldots, c_n$, where $c_i \in X = \{0,1\}$; PRBG seed; and a constant, k

- **Output:** The compressed binary file.
  - $j = 1$, initial distribution, D, is uniform, $p(x_i) = 1/n$, $1 \leq i \leq n$. Here $n=2$.
  - Generate the next pseudo-random bit b using the PRBG.

while j ≤ n **do**
    Using k_b, find the model to follow.
    Read a bit, c_j, from the input file, and find the range where the message would lie.
    If any symbol can be identified clearly from that range
        Append that to the output file.
        t=t+1
        Update the probabilities using the Laplace formula.
    Else Proceed to read the next bit, c_j.
If $t==k$
    Generate the next pseudo-random bit b using the PRBG.
    $t=0$;j = j + 1
end if
end while

Decoding Example

Let us take the same example as encoding, and we assume that the decoder has the bits 101101011 as input. The initial assumptions about the ordering and symbol set are the same as that of the encoding example. Initial probabilities of the two codewords are uniform, i.e., 0.5 each. Decoding proceeds as follows:

i) The first bit of PRBG is 1, so the order to be followed is as per Figure 4.2(a). The first bit of the codeword is observed to be 1, so the decoder knows the range of the message would lie above 0.5, and (0.5,1] is the selected range as shown in Figure 4.10. This range implies the input would be 1.

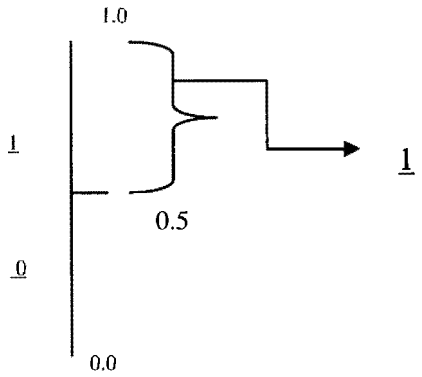

Fig4.10. Representation of the symbols in the range (0,1] in ratio of their probabilities.

While decoding every time, the decoder would divide the selected range in half and consider the lower half to be 0 and the upper half to be 1. For example, the range (0,1] is divided, and any range below 0.5 is considered to be 0 while that on the upper half is considered to be 1. Because the first bit encountered by the decoder is 1, the decoder would know that the range in which the message would lie should be (0.5,1]. This range corresponds to the symbol 1, so it outputs 1.

ii) At the next stage, the probabilities of the input gets updated using the Laplace formula, and becomes $p(1) = 0.66$ and $p(0) = 0.33$. The selected range is divided into the ratio of their probabilities leading to Figure 4.11

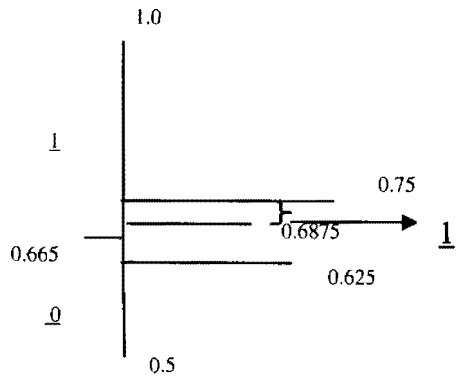

Fig4.11. Representation of the symbols in the range (0.5,1] in ratio of their probabilities.

The range (0.5,1] is again divided in half, and any range below 0.75 would be 0 and above would be 1. The next bit in the codeword is 0 which implies that the message would lie in the range (0.5.0.75], but the ranges of 0 and 1 are (0.5,0.665] and (0.665,1], respectively, so the range (0.5,0.75] does not lead us to any symbol. Therefore, the decoder proceeds and reads the next bit in the codeword. The next bit is 1 which implies the range in which the message lies would be (0.625,0.75]. Even this range does not facilitate a guess for the next symbol as it does not completely lie in any of the ranges. The next bit read by the decoder is 1 which implies that the range in which the message lies would be (0.6875,0.75]. From Figure 4.11, we can see that this range is a part of the actual range (0.665,1] which implies the second symbol given by the decoder would $\underline{1}$.

iii) The probabilities of the symbols get updated after a symbol is output by the decoder. The updated probabilities are $p(\underline{1}) = 0.75$ and $p(\underline{0}) = 0.25$. Because the last symbol was $\underline{1}$, we would divide the range (0.665,1] in the ratio of the updated probabilities. This leads to Figure 4.12 The next bit is 0, meaning that the message would lie below 0.71875; i.e., the range would be (0.6875,0.71875]. From Figure 4.12 we can clearly say that the range mentioned corresponds to the symbol 0.

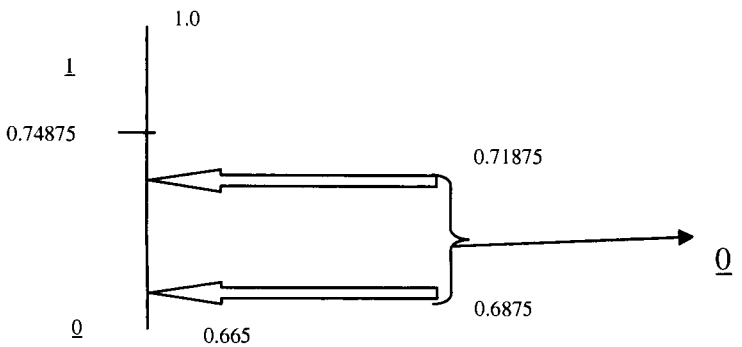

Fig 4.12. Representation of the symbols in the range (0.665,1] in ratio of their probabilities.

iv) We know that our key, k, is equal to 3 in this example. This means that, after every 3 symbols, the order of the symbols in the range should switch depending on the PRBG bit output. Because 3 symbols have been decoded, it is time to switch the order. The next PRBG bit is 0, so the order should be according to Figure 4.2(b) The arrangement of the symbols in the new range (0.665,0.74875] is as follows. We already know from the previous step that the message lies in the range (0.6875,0.71875]. Now, we divide this range and consider the next bit. The next bit is 1, so the range should be above 0.703125, i.e., (0.703125,0.71875]. This does not help the decoder to emit a symbol, so it divides this range again and reads the next bit available to it. The next bit is 0, so the range should be on the lower half of 0.7109375. From Figure 4.13 it can be clearly stated that the range (0.703125, 0.7109375] shows the next symbol to be 1.

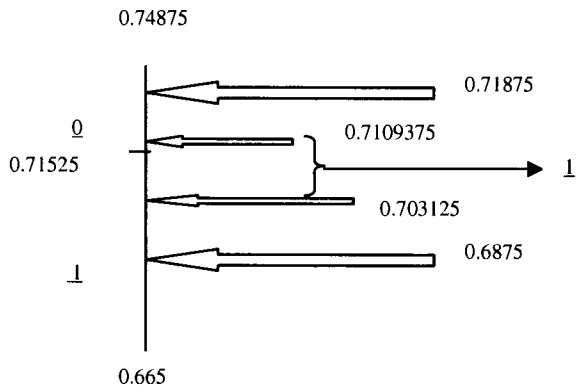

Fig4.13. Representation of the symbols in the range (0.665,.74875] in ratio of their probabilities.

Updating the probabilities and reading the next two bits available to the decoder,

the next symbols are found as described in Figures 4.13 and 4.14. From the previous step, we know that the range in which the message would lie is (0.703125,0.7109375]. After we divide the range (0.665, 0.71525] into a ratio of the probabilities of the symbols, the decoder immediately gives a symbol as the range in Figure 4.14 clearly shows that the next symbol would be 0 .

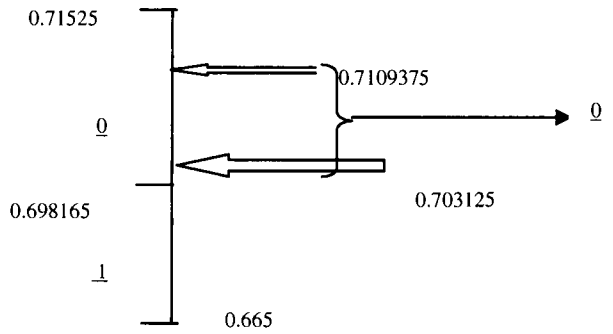

Fig4.14. Representation of the symbols in the range (0.665,.71525] in ratio of their probabilities.

vi) The probabilities are updated, and the next bit scanned by the decoder is 1. Figure 4.15 shows how the decoder determines the next symbol.

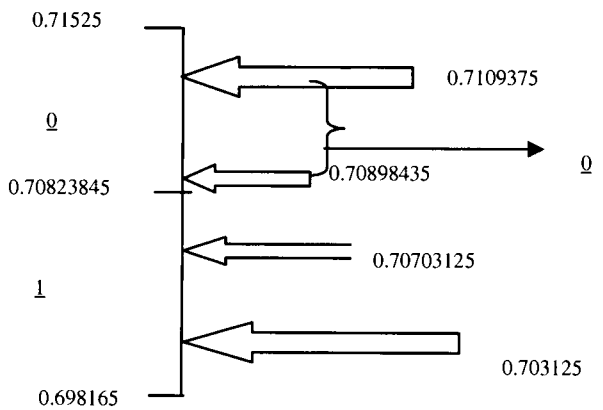

Fig4.15. Representation of the symbols in the range (0.698165,.71525] in ratio of their probabilities.

4.3. Cryptanalysis of Modified Arithmetic Coding

The algorithm proposed in the previous section has used an output of a secure PRBG as its key. A pseudorandom bit generator is considered to be secure if it satisfies the next bit test which is one of the fundamental tests of pseudorandomness. The next bit test states that, given the first k bits of a random sequence, there is no polynomial time algorithm that can predict the $(k+1)th$ bit with a probability of success higher than 50%. The main objective of this section of the chapter is to prove the security of the proposed method. We do so by a method of contradiction.

Let us assume that the new modified Arithmetic coding proposed is not secure. Let us assume the attacker has in hand an algorithm, A, which can produce the corresponding message (without the knowledge of the key) given an encoded string as input with a success probability of $\frac{1}{2}+\epsilon$, where ϵ is very small. The attacker has access to the oracle, A, and an encoded string, C. The attacker is also able to get hold of a few bits of the output of the same PRBG which the encoder used during encoding. Let the encoded sequence be $C=c_1c_2c_3....c_nc_{n+1}...c_{n+k}$, where c_i denotes the binary bits of the string. Let $B=b_1b_2...b_l$ be the first l bits of a secure PRBG. Suppose $X = x_1x_2... x_t...x_m$ be a message available to an attacker and an assumption is made that $t=kl$. We know after every k symbols the encoder would switch the order of the symbols depending on the PRBG bit. Since $t=kl$, this implies that to encode t symbols of the message X, l bits of B would be needed. The binary bit in B determines which order of symbols (4.2(a) or 4.2(b)) follows. Suppose feeding the first C and using the l bits of B in the algorithm A the attacker gets the message as $X= x_1x_2... x_t$. Algortihm 6 is developed using Algorithm A

Algorithm 6. Algorithm showing an attack on Modified Arithmetic coding

Assumption: The attacker knows the quantity, k. PRBG is secure.
Input: Compressed sequence $C = c_1c_2c_3....c_nc_{n+1}...c_{n+k}$.

- Sequence B containing the first l bits of the PRBG. B $=b_1b_2...b_l$.
- oracle of Algorithm A.

Output: The $(l+1)th$ bit of B.

- Call $A(C)$, and get the output as $X = x_1x_2...x_m$, where $m>n+k$. X is the message corresponding to C. Here, we assume that the first n bits of C are decoded using the first l bits of B to give supposed t bits of X, but the next bits of B are not known to the attacker. $t=kl$
- Let us just pick the first n bits of C to get $C_1 = c_1c_2c_3....c_n$ and feed it to Algorithm A. Let the output be $X_1 = x_1x_2...x_t$. Since $t=kl$, C_1 is decoded using l bits of B.
- Because $t=kl$, therefore, at this point, there should be a switch in the order depending on the P.
- Let us build a sequence, $B' = b_1b_2...b_l0$, which means the $(l+1)th$ bit of B' is 0.
- Call decoder $D(C,B')$, and get message X'. This implies that the decoder used the order 0 [i.e., (ii) of Figure 1] for decoding bits $c_{n+1}...c_{n+k}$.
- Let $B'' = b_1b_2...b_l1$ which means the $(l+1)th$ bit of B'' is 0.Call decoder $D(C,B'')$, and get message X''. This implies that the decoder used the order 1 [i.e., Figure 4.2(a)] for decoding bits $c_{n+1}...c_{n+k}$.
- If $X==X'$, then the next bit of B is 0; else if $X==X''$, then the next bit is 1.

We observe in Algorithm 6 that after decoding the first t symbols the attacker does not have any pseudorandom bit left and therefore is not able to guess which order to use for decoding the next few bits of the codeword. He builds up two bit string B' and B'' as shown in the algorithm. Using the next bit of B' and B'' he decodes the rest of the message. He gets two messages X' and X''. Now, comparing these two messages with the actual message from the output of Algorithm A the attacker is able to guess which bit string B' or B'' is used. This makes it possible to guess the next bit of the PRBG output.

Therefore, we could see that Algorithm 6 is able to produce the next bit of the PRBG. The success probability of A to produce the right message, taking C as input, is $\frac{1}{2}+\epsilon$. Given the encoded sequence and the key as input, decoder D would produce the correct message with a success probability of 1. Because *A* and *D* are independent of each other, the success probability of Algorithm 6 to produce the *(l+1)*th bit is $(\frac{1}{2}+\epsilon).1=\frac{1}{2}+\epsilon$. This proves that PRBG *B* is failing Next bit test and is not secure. We assumed the PRBG to be secure so it should not fail next bit test i.e. there is no polynomial time algorithm which could produce the *(l+1)th* bit of PRBG knowing sequence *B* with a success probability more than $\frac{1}{2}$. Therefore, there is no such algorithm as Algorithm 6. Since Algorithm 6 is built with the assumption that there exists an algorithm, *A*, it can be further concluded that there does not exist any algorithm, *A*, which is able to decode the sequence encoded by Modified Arithmetic Coding without knowledge of the key. Hence, Modified Arithmetic Coding is secure. The attack developed in Algorithm 6 takes into account that the attacker is aware of the value *k* but not the seed of the PRBG. Even knowing the key *k* the attacker has very low success probability of breaking the codeword. Therefore we can conclude If an attacker is not aware of the key *k* Algorithm 6 becomes more complex thereby adding more security to the encoding and decoding techniques.

Example of an attack

Let us take the above decoding example in section 4.2 to see how the attacking algorithm works. As we assumed that there exist an algorithm *A* which would output the message with a success probability of $\frac{1}{2}+\epsilon$ taking as input just an encoded string. We take

the example shown in section 4.2 . Let us input the codeword C = 101101011 in the algorithm A and gets the message X= 1 1 0 1 0 0.

Let us follow the Algortihm 6 to attack the encoding process. The first bit of the PRBG is 1. We assumed the attacker knows that bit which means $B = 1$ and therefore $l=1$. Since $k=3$ and $l= 1$ and we know $t = kl$, this leads to the fact that $t=3$. From decoding example in section 4.2 we know that to output the first 3 bits of the message the decoder would need first 5 bits from the encoded message . This means for this example $n= 5$ and C_1=10110 , X_1 = 1 1 0 and D(C_1,B) = X_1 . At this point since three symbols are already decoded by the decoder so, now it's the time to read the next bit of PRBG and decide on the order of the symbols in a range. But the next bit of the PRBG is unknown to attacker. So , the attacker guesses the next bit of the PRBG output and builds up two possible bit strings $B'=11$ and $B''=10$. The attacker calls the decoder once again and feeds the whole message with these new B' and B'' respectively. $D(C,B')=X'$ and $D(C,B'')=X''$ where X'= 1 1 0 1 0 1 and X'' = 1 1 0 1 0 0. Comparing X' *and* X'' with X we find that $X=X''$ and so the PRBG bit used should be B'' which means the next bit of the PRBG would be 0. Thus the attacker is able to guess the next bit of the PRBG successfully. This shows we are able to build an algorithm which could make the secure PRBG fail the next bit test. However this conclusion does not stand as we know that the pseudorandom bit generator taken into account in this example is secure , so there cannot be any algorithm as Algorithm 6. Since Algorithm 6 is built with the assumption that there exists an algorithm, A, it can be further concluded that there does not exist any algorithm, A, which is able to decode the sequence encoded by Modified Arithmetic Coding without knowledge of the key. Hence, Modified Arithmetic Coding is secure.

Thus it is proved under the assumption that if there exist an algorithm A and attacker has access to it then the proposed Modified Arithmetic coding is secure.

CHAPTER 5. CONCLUSION

Thus, it can be concluded that the two proposed compression algorithms, Adaptive SFE and Modified Arithmetic Coding, can also be used for encryption. The two algorithms are developed incorporating little pseudorandomness in the algorithm. These pseudorandomness facilitated to use the compression algorithms for encryption too. Following is a summary of the new Adaptive SFE.

i) Adaptive SFE uses two orderings, K_0 and K_1, one of which is chosen for the compression based on the output of a pseudorandom bit sequence generator.

ii) The probability mass function, D, used in the encoding is updated every t steps according to the frequency of symbols observed so far.

The compression properties of the new algorithm are virtually the same as the simple SFE compression algorithm. However, the complexity of the attack increases dramatically to $2^{m/10\frac{n!(n!-1)}{2}}$, where m is the size of the file being compressed and n is the number of symbols in the alphabet set. The new adaptive SFE method requires probabilities to be computed nm/t times during a compression. This minor increase in computation allows the method to compress sources with unknown probability mass functions.

The attacking algorithm becomes quite complex for Adaptive SFE keeping the compression ratio almost similar to Normal SFE. Similarly, Modified Arithmetic Coding also uses the same Arithmetic coding technique, thereby not affecting its compression

capabilities. However, Modified Arithmetic Coding has security incorporated in it which makes it a very useful compression technique over simple Adaptive Arithmetic coding.

This thesis also leaves scope for few future work like the Arithmetic coding could be made more secure so that it can withstand other strong attacks like chosen plaintext , Known plaintext etc. Moreover the examples and algorithms for Modified Arithmetic coding are described taken into account a binary symbol set , future research work could be extend the same algorithm to M- ary symbol set and study the security and compression capabilities of the same.

REFERENCES

[1] C. –P. Wu and C.-C. J.Kuo," Design of integrated multimedia compression and encryption schemes," *IEEE Trans. Multimedia,* vol. 7, no. 5, pp. 828-839, Oct 2005.

[2] D. W. Gillman, M. Mohtashemi, and R. L. Rivest, "On breaking a Huffman code," *IEEE Trans. Inform. Theory*, vol. 42, no. 3, pp. 972-976, May 1996.

[3] X. Ruan and R. S. Katti, "A new source coding scheme with small expected length and its application to simple data encryption," *IEEE Transaction on Computers*, vol 55, Issue 10, pp. 1300-1305, Oct. 2006.

[4] Thomas M. Cover and Joy A. Thomas, "Elements of Information Theory", *John Wiley & Sons*, 1991.

[5] J.Katz and Y.Lindell, *Introduction to Modern Cryptography,* Chapman & Hall/CRC, 2008.

[6] Xiaoyu Ruan and Rajendra Katti, "Using Improved Shannon-Fano-Elias Codes for Data Encryption," *2006 International Symposium on Information Theory*, pp 1249-1252.

[7] D.J.C Mackay,"Information Theory, Inference and LearningAlgorithms," *Cambridge University Press*,2003.

[8] C. Finnila, "Combining data compression and encryption" in WESCON/94, *"Idea/Microelectronics" Conf.* Rec., Anaheim, CA, pp-404-409, Sep. 27-29, 1994.

[9] C.-E Wang,"Simultaneous data compression and encryption" *Code Breakers Journal* vol 2 no 3 2005.

[10] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. Information Theory, vol 23*, issue 3, pp. 337-343, May 1977.

[11] C-P. Wu and C. Kuo, "Efficient multimedia encryption via entropy codec design," *Proceedings of SPIE, vol.4314,* Jan. 2001.

[12] Ranjan Bose and Saumitr Pathak, "A Novel Compression and Encryption Scheme Using Variable Model Arithmetic Coding and Coupled Chaotic System" IEEE Trans on Circuits and Systems, vol. 53 no. 4 April 2006.

[13] J.Wen, H.Kim and J.D Villasenor, "Binary arithmetic coding with key-based interval splitting," *IEEE Signal Process.Lett.,* vol. 13, no.2, pp. 69-72,Feb 2006.

[14] H.Kim, J.Wen and J.D. Villasenor, "Secure Arithmetic coding," *IEEE Trans. Signal Processing,* vol.555, no. 5, pp. 2263-2272, May 2007.

[15] J.G. Clearly, S.A.Irvine, and I. Rinsma-Melchert, "On the insecurity of arithmetic coding," *Comput. Security,* vol. 14, pp. 167-180, 1995.

[16] M. Grangetto, E. Magli, and G. Olmo, "Multimedia selective encryption by means of randomized arithmetic coding," *IEEE Trans. on Multimedia, vol 8. no. 5*, Feb. 2006.

[17] M. Grangetto, A. Grosso, and E. Magli, "Selective encryption of jpeg 2000 images by means of randomized arithmetic coding".

[18] R. Norcean and A.Uhl, "Selective encryption of the JPEG2000 bitstream," *Lecture Notes in Computer Science,* 2828, 2003, pp. 194-204.

[19] C.Shi and B.Bhargava, "A fast MPEG video encryption algortithm," in *Proc.6th International Multimedia Conference,* Bristol,U.K.,Sept. 12-16, 1998.

[20] C.Shi, S.-Y. Wang and B.Bhargava, "MPEG video encryption in real time using secret key cryptography," in *1999 Int. Conf. Parallel and Distributed Processing Techniques and Applications(PDPTA '99),* Las Vegas, NV, Jun-July 28-1,1999.

[21] L.Tang, "Methods of encrypting and decrypting MPEG video data efficiently," *Proc. 4th ACM Int. Multimedia Conf.,* Boston , MA Nov. 18-21, 1996, pp. 219-230.

[22] M. A. Haleem, C. N. Mathur, and K. P. Subhalaksmi, "Joint distributed compression and encryption of co related data in sensor networks,"*MILCOM,* 2006.