Hindawi Publishing Corporation
EURASIP Journal on Information Security
Volume 2010, Article ID 621521, 9 pages
doi:10.1155/2010/621521

Research Article

Secure Arithmetic Coding with Error Detection Capability

Mahnaz Sinaie and Vahid Tabataba Vakili

Department of Electrical Engineering, Iran University of Science and Technology, Narmak, Tehran 1684613114, Iran

Correspondence should be addressed to Mahnaz Sinaie, msinaie@ee.iust.ac.ir

Received 9 February 2010; Revised 23 May 2010; Accepted 7 September 2010

Academic Editor: Enrico Magli

Copyright © 2010 M. Sinaie and V. T. Vakili. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Recently, arithmetic coding has attracted the attention of many scholars because of its high compression capability. Accordingly, this paper proposed a Joint Source-Cryptographic-Channel Coding (JSCC) based on Arithmetic Coding (AC). For this purpose, embedded error detection arithmetic coding, which is known as continuous error detection (CED), is used. In our proposed method, a random length of forbidden symbol which is produced with a key is used in each recursion. The dummy symbol is divided into two dummy symbols with a key and then is placed in random positions in order to provide security. Finally, in addition to producing secure codes, the suggested method reduced the added redundancy to half of the total redundancy added by CED. It has less complexity than cascades source, channel coding, and encryption while its key space in comparison to other joint methods has enlarged. Moreover, the coder provides a flexible switch between a standard compression model and a joint model.

1. Introduction

The increasing demand for the use of computer networks, the wide availability of digital multimedia contents, and the accelerated growth of wired and wireless communications have resulted in new research areas in joint coders.

The design of modern multimedia communication systems is very challenging as the system must satisfy several contrasting requirements [1]. Data compression is needed because it provides a mechanism to increase the effective bandwidth in a network and serves the highest possible number of users. Data compression optimizes the required storage space and reduces transmission time in the network. In one hand, compression typically makes the transmission very sensitive to error or packet losses, thus it can decrease the quality of received data by the final users so channel coding is required for error detection and correction [2]. On the other hand, source coding decreases redundancy in the plaintext which makes the data more resistant to statistical methods of cryptanalysis [3], and additionally, the accessibility of data makes it possible for the unauthorized users to reach the data easily. Therefore, to be reliably and confidentially transmitted, the data must be encrypted [4].

Many data compression techniques are available for efficient source coding [5, 6]. Strong error control codes have been developed for channel coding. In addition, some encryption algorithms have been developed for secure data transmission. Recent source coding, channel coding, and encryption algorithms require computational power for encoding and decoding. This is particularly unfavorable in certain applications such as mobile communications, embedded systems and real-time communication, where devices (e.g., portable equipments) are resource constrained due to the size limitation and power consumption considerations [2].

In real-time or satellite communication, delay and complexity are not desirable. Therefore, low complexity JSCC is preferable for such situations. Techniques for joint source-channel coding, which have been proposed in this research, use the duality of source encoding and channel decoding and are aimed at decoding noisy compressed data as reliably as possible. The development of these joint algorithms has closely followed the development of source and channel coding algorithms.

Most of the early works on joint source-channel coding used different forms of Huffman codes. But nowadays, by increasing interest in arithmetic coding in the multimedia

applications [7], for example, JPEG2000 and H.264, many researchers were attracted to it. In 1997 Boyd et al. [8] introduced a forbidden symbol in the source alphabet and used it at the decoder side as an error detection device. Sayir [9] considered the arithmetic coder as a channel coder and added redundancy in the transmitted bit stream by introducing gaps in the coding space and shrinking the probability of symbols by a factor [10]. In these joint coders, we have embedded error detection compressed data without providing essentially any security in the face of a chosen plaintext attack, in which an attacker has the ability to specify a sequence of input symbols, to observe the corresponding output, and to repeat this process for an arbitrary number of times.

Some schemes of joint AC and encryption have been also proposed up to now. Wen et al. [11] modified the traditional AC by removing the constraint that intervals corresponding to each symbol are continuous and the intervals associated with each symbol can be split according to a key which is known for the both encoder and decoder. Grangetto et al. [12] proposed a method in which the system modified the traditional arithmetic coder by randomly permuting the intervals in accordance with a key.

Magli et al. [1] developed a JSCC. It used arithmetic coding which was proposed by Sayir and for providing security; it randomly permuted the intervals in accordance with a key generating shuffling sequence which was introduced by Grangetto. Although this system is a JSCC but the attacker can break the system by comparing N pairs of the output with the corresponding input which differ from each other in exactly one symbol. Teekaput and Chokchaitam [13] have introduced a scheme for JSCC. Security was provided by changing the location of the forbidden symbol. This system looks like the system which was introduced by Magli et al., so it suffered from the same limitations.

In this paper, we present a method for joint source-cryptographic-channel coding based on arithmetic coding. This is very important in light of simplifying the design of the system. We use binary arithmetic coding with the forbidden symbols which was introduced in [14] for error detection. Security is provided by using random length of the forbidden symbols and randomly placing these dummy symbols in the probability table. Compression ratio is improved in comparison with the systems in [1, 13]. Also, the actual key space has enlarged. This method can be used for arithmetic coding with multiple symbols. However, to simplify the method, we use binary AC.

The rest of this paper is organized as follows: in Section 2, we discuss more on arithmetic coding and arithmetic coding with forbidden symbol. In Section 3, our proposed method for JSCC is described. In Section 4, the results obtained from the simulation and the performance of the system are explained. In Section 5, we draw some conclusions.

2. Arithmetic Coding and CED

This section provides a brief introduction to arithmetic coding and AC with forbidden symbol which is named in [14] as CED. Until AC was developed in the 1970s, Huffman coding was considered to be almost optimal. Huffman coding uses a tree for encoding a sequence. AC uses a one-dimensional table of probabilities instead of a tree. It always encodes the whole massage at once and allows the allocation of fractional number of bits to each source symbol. It generates a code sequence which is uniquely decodable, such that the probability of distribution of code sequence approaches the uniform distribution over the code alphabet [6].

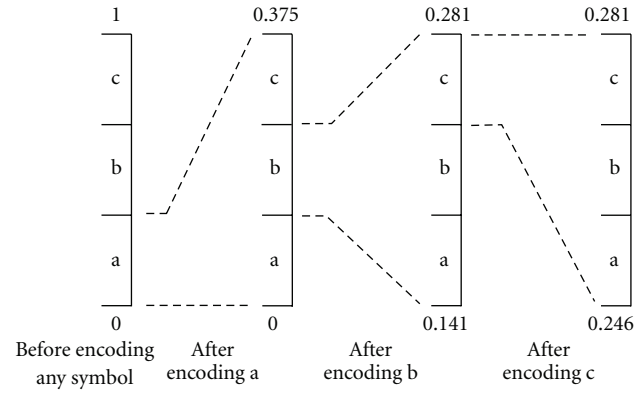

Figure 1: An example of arithmetic coding, the source symbols are a, b, c with $p(\mathrm{a}) = 0.011$, $p(\mathrm{b}) = 0.011$, $p(\mathrm{c}) = 0.010$ [10].

AC works by recursively subdivision of coding interval in portion to probabilistic estimates of symbols as generated by a given model and retains it to be used as the new interval for the next encoding step of the recursion [5]. This can be illustrated better with an example. Consider a source alphabet with three symbols [14] a, b, and c with $p(\mathrm{a}) = 0.375$, $p(\mathrm{b}) = 0.375$, and $p(\mathrm{c}) = 0.25$. For example, we want to encode the sequence abc. After encoding a, the new interval will be $[0,\ 0.375)$, and the transmitted sequence would lie in this interval. The next symbol is b, and according to the intervals associated to each symbol, the next interval will be $[0.141,\ 0.281)$. This recursion continues to the end of the sequence. At the end, a number in the last interval which is a fractional number between zero and one will be sent as the sequence code. This example is illustrated in Figure 1.

AC is a powerful source coding technique and has higher compression efficiency than other entropy coders. But arithmetic coding has two major drawbacks, the error sensitivity and error propagation property. Error propagation because of loss of synchronization can damage the whole data after an error has occurred in the compressed data. We can use this loss of synchronization to error detection. Anand et al. in [14] introduced a forbidden symbol which does not belong to the source alphabet and never occurs in the probability table. For inserting dummy symbol into the probability table, probability of the symbols should be shrunk by a factor. This forbidden symbol has a finite and small probability assigned to it. If its probability is ε, so the probability of the symbols must be shrunk by factor $(1 - \varepsilon)$. The introduction of the forbidden symbol produces an amount of artificial coding redundancy per encoded bit equal to

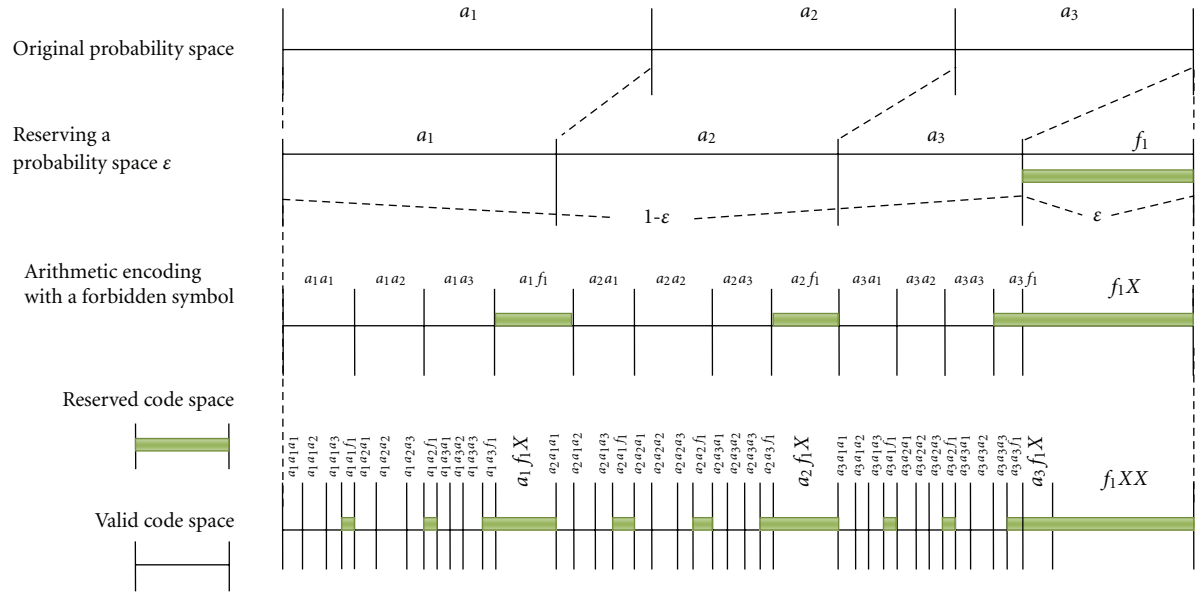

Figure 2: Encoding with a forbidden symbol for probability ε.

$-\log_2(1-\varepsilon)$, at the expense of the compression efficiency [14]. The decoder obtains an error detection capability and enhances its robustness against noise. If an error occurs, this forbidden symbol is very likely to be eventually decoded with a high probability. Figure 2 illustrates a sample of binary AC subinterval separation by inserting a forbidden symbol in the current interval.

This forbidden symbol can be placed anywhere in the probability table, and we can also have more than one forbidden symbol and place them in more than one location in the probability table. In conventional CED, the probability of the forbidden symbol is fixed, and also the forbidden symbol is fixed at the same location for the whole encoding process. Before transmission, the encoder and decoder should negotiate the location and size of the forbidden symbol [13]. If its probability is fixed for the whole encoding process, then the bit rate of the code is fixed, and the amount of the added redundancy is fixed to $-\log_2(1-\varepsilon)$ bit per symbol.

If we take the maximum bit rate needed into account and also consider that the bit rate in each recursion is not allowed to exceed the maximum bit rate, we can change the bit rate while encoding. This causes less redundancy to be added to the bit stream and higher security. We describe this in more details in Section 3.

3. Scheme of the Proposed Model

The present paper aims to provide an arithmetic coding system which is secure and has an error detection capability. Our scheme is based on CED, in which there is a forbidden region with a probability ε, added to the probability table to provide some redundancy while a synchronized decoder can detect the error occurring and conceal wrong decision bits. The combined data encryption and AC use the error propagation property of AC to provide security. Our proposed technique uses forbidden symbols with random lengths and places them in random locations. The flowchart of this proposed technique is shown in Figure 3. While the concept of this scheme can be applied to a source alphabet with any size, for simplicity, the remainder of the discussion focuses on the binary case.

3.1. Inserting Forbidden Symbols. In conventional CED, the probability of the forbidden symbol is fixed and at the beginning of the encoding process; this probability which is named ε is determined by (1). This depends on the maximum bit rate, R and the entropy of source, $H(A)$ [9]:

$$\varepsilon = 1 - \left(\frac{1}{2}\right)^{(1/R-1)H(A)}. \tag{1}$$

Adding the forbidden symbol leads to the addition of redundancy to the output extension which can be used as a means of error detection. This method does not have enough security against attacks; therefore, we use a random-length forbidden symbol in each recursion in our scheme instead of a fixed-length one. In each recursion with a random generator, we generate a forbidden symbol in the range $[0,\ \varepsilon)$, in which ε is determined by maximum bit rate, R, by using (1). The generated probability of the forbidden symbol in each recursion is named γ. By using this random forbidden symbol in every recursion, we shrink the probability of symbols by the factor $(1-\gamma)$. This causes adding random redundancy while encoding each input symbol. In addition, we can claim that we have a semiadaptive arithmetic coder because in each recursion, a different length of the forbidden symbol is produced. It leads to a different shrinking factor

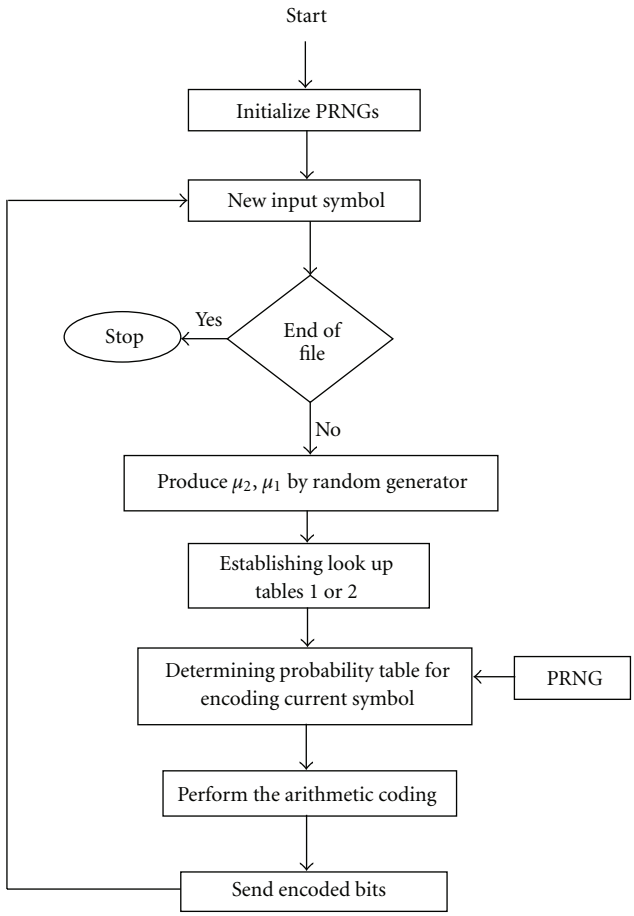

Figure 3: Flowchart of the proposed scheme.

in each recursion. Therefore, the probability of the source symbols with various factors would be shrunk.

In the previous section we said that we can have more than one forbidden symbol, therefore we use two forbidden symbols in this method. Since the sum of the probabilities of two forbidden symbols must be equal to γ, we can divide the generated forbidden symbol, μ, in each recursion equally, or generate another forbidden symbol in the range $[0, \gamma)$ and then uniformly divide the forbidden symbol to two forbidden symbols μ_1, μ_2 with probabilities of γ_1 and γ_2.

The γ_1 and γ_2 represent the encryption key which is also referred to as **K** in the following sections and adjusted with a proper precision in an acceptable range depending on the requirements of different applications. At the decoder side, if a synchronized decoder is applied, that is, adding the γ_1 and γ_2 at each coding step, data will be reconstructed accurately. Otherwise, whether using a standard AC decoder or a decoder of proposed scheme with a different γ_1 and γ_2, the encoded code stream cannot be correctly decoded.

3.2. Establishing and Selecting the Probability Table. In Section 2 we demonstrated that the forbidden symbol can be placed anywhere in the probability table. In binary AC, it can be placed at the beginning, in the middle and at the end of the probability table. We use Pseudorandom Number Generator (PRNG) to control the place of the forbidden symbols. *A* seed value, *S*, which also represents another encryption key, is used to initialize the PRNG. The bits of the generated random sequence are used as an encryption key in each recursion. In practice, the random sequence is taken on the values 0 and 1 with probability of .5 which is also the controlling bits sequence.

Table 1: Mapping function of binary arithmetic codes with two different lengths of forbidden symbols (look up table).

Situations	Situations
(**a b** $\boldsymbol{\mu_1\mu_2}$)	($\mu_1\mu_2$a b)
(**a** $\boldsymbol{\mu_1}$**b** $\boldsymbol{\mu_2}$)	(μ_2a μ_1b)
($\boldsymbol{\mu_1}$**a b** $\boldsymbol{\mu_2}$)	(a μ_2b μ_1)
($\boldsymbol{\mu_1}$**a** $\boldsymbol{\mu_2}$ **b**)	(μ_2 a b μ_1)

Table 2: Mapping function of binary arithmetic codes with two equal lengths of forbidden symbol (look up table).

Situations
(**a b** $\boldsymbol{\mu_1\mu_2}$)
(**a** $\boldsymbol{\mu_1}$**b** $\boldsymbol{\mu_2}$)
($\boldsymbol{\mu_1}$**a b** $\boldsymbol{\mu_2}$)
($\boldsymbol{\mu_1}$**a** $\boldsymbol{\mu_2}$**b**)

If we divide forbidden symbol μ unequally, μ_1 and μ_2 are in different ranges so a binary memoryless source, **X**, with probabilities P_0 and P_1 is encoded by means of a quadruplet AC with the alphabets a, b, μ_1, μ_2. For allocating these, we have different possibilities which are demonstrated in Table 1. In this situation, for encoding each input symbol, we use 3 bits of the generated random sequence of PRNG as a key to control the locations of the forbidden symbols. But, if we divide μ equally, we will have ternary AC, and Table 2 is its look up table. This look up table uses 2 bits for determining the locations of the forbidden symbols.

To conclude, we do not encrypt the code string which causes a totally different value but only secretly add subintervals and secretly place them. The proposed encoder works with a key $\mathbf{K} = (\gamma_1, \gamma_2, S)$, which represents the final encryption key. Given the same **K**, both the encoder and the decoder generate the same pseudorandom number sequence for decision bits and exactly add the same γ_1, γ_2 to the corresponding code string in order to synchronize them with each other. On the other hand, no matter which parameter of **K** is unknown or incorrectly given, the decoder cannot decode the compressed data properly, and the decompressed data is almost meaningless. Furthermore, as long as γ_1 and γ_2 are set to 0, our scheme achieves a simple switch from the joint compression, error detection, and encryption model to a standard compression model. Also by setting the sum of γ_1 and γ_2 equal to ε, this JSCC is transformed to joint compression and error detection. Thus, this scheme can be used for selective encryption and apply to portions of data which needs more security. Nevertheless, an efficient and

secure key distribution protocol is one of the challenging issues and is beyond the scope of this paper.

4. Simulation Results

Our proposed scheme has been implemented with Matlab software and a personal computer with 2 G of RAM and Intel Centrino Core 2 Duo 2.2 G as its CPU. Due to unstable possesses in computer systems, we take 20 trials and select the most frequently occurred results as the final values. Input symbols, upper and lower bounds, and also produced forbidden symbols in each recursion are set with precisions of 10^{-6} being equal to 16-bit implementation. It is worth noting that this precision is not fixed and can be flexibly adjusted depending on the requirement of the target applications.

4.1. Compression Ratio. The Joint Source-Cryptographic-Channel Model should be used with the precondition that there is no large redundancy generation after modifying the standard coding engine. Table 3 shows the results of applying the proposed method for input sequences with lengths of 100, 1000, and 10000 symbols and allows for comparison with traditional arithmetic coding in absolute as well as relative terms. The upper half of the table considers the case where $p(\mathrm{a}) = 1/3$, and the lower half of the table considers the case where $p(\mathrm{b}) = 5/6$. The exact length of the output depends not only on the input data but also on the specific sequence of forbidden symbols, as well as their locations and lengths in each recursion. Therefore, the code lengths shown in the table are averages based on simulations using 1000 random sequence realizations. The column labeled "proposed method" gives the mean of the code lengths based on a large number of simulations using random seeds for location and lengths. These results show that in order to limit coding redundancy, ε should be defined in a limited range, which can be flexibly controlled according to the requirement of various application systems. Table 3 shows that the redundancy added to the bit stream by our model is half of the redundancy which conventional method adds to the bit stream. For $\varepsilon = 0.03$, redundancy is 0.0439 bit per symbol. If the length of ε is fixed, for example, $N = 100$, the redundancy which is added is 4.39 bit per symbol, but our proposed method adds 2.1 bit per symbol.

Using the forbidden symbol in the source alphabet actually aims at simply detecting errors and not correcting them. By randomizing the forbidden symbol, although the amount of the added redundancy is reduced to half, this does not interfere with the capability of error detection. However, Anand et al. [14] gave an empirical model to estimate the number of the bits necessary to detect an error after it has occurred. This is shown in the following:

$$P_y(k) = (1-\varepsilon)^{k-1}\varepsilon, \quad k = 1, 2, \ldots, \infty. \tag{2}$$

The probability of not detecting an error after n bits is

$$P(n) = (1-\varepsilon)^n. \tag{3}$$

Based on the extensive performed simulations, it is concluded that in the CED method, if n bits are needed for detecting an error after it has occurred, $(6 * n)/5$ bits are needed for error detection in our proposed method. Hence, to solve this problem, we can compensate for this shortcoming by assuming greater lengths for the input blocks in the proposed encoder. However, we know that adding security and error detection capability to a compression encoder often leads to a compromise between the amounts of compression achieved and the amount of security and the robustness against channel errors incorporated.

The encoded stream can be reconstructed perfectly by providing the same **K** and by reversing the encoding operations. By having the same **K,** both encoder and decoder generate the same pseudorandom number sequence for decision bits and exactly add the same γ_1 and γ_2 to the corresponding code string in order to synchronize with each other. As soon as the forbidden symbol is decoded, the occurrence of error in the received sequence is detected. However, this method of decoding is not capable of correcting the errors. But, the redundancy of the encoder's output can be used for correcting errors.

Arithmetic codes can be viewed as tree codes. Sequential decoding is a general decoding algorithm for tree codes. It was introduced by Wozencraft and Reiffen to decode convolutional codes in [15]. Fano [16] presented an improved sequential algorithm in 1963, which is now known as the Fano algorithm. Pettijohn et al. [17, 18] proposed two sequential decoding algorithms, depth first and breadth first, for decoding arithmetic codes in the presence of channel errors. We can use these decoding algorithms with the same key for decoding the output of our proposed scheme.

4.2. Complexity. Sayir [10] showed that an arithmetic coder can be an entropy source encoder when the model is matched with the source and can be a channel encoder when the probability space is properly reserved for error protection and can act as a convolutional code. After inserting the forbidden symbol to a source with M alphabet, we will have an arithmetic coding with M + 1 alphabet in which one of the symbols never appears. Therefore, adding parity is performed while compression without adding more additional operations to the conventional arithmetic coding. If the source has M alphabet, so this method just adds M multiplication and 1 additional operation to the complexity of conventional arithmetic encoder. But if we want to place a convolutional encoder after arithmetic encoder, according to the amount of redundancy, it needs some shift and XOR operations and increasing memory usage. For example, if the bit rate is 1/2 and the code generator polynomial is $p(x) = x^2 + x$, it would need at least three shift register and XOR operations for each input symbol.

Also because a traditional arithmetic coder needs to work sequentially, arithmetic coding and convolutional coding cannot be parallelized. A comparison of time duration for arithmetic coding and arithmetic coding followed by a 1/2 feedforward convolutional encoder is shown in Figure 4.

Placing the forbidden symbol in different locations and assigning random lengths of the forbidden symbols

Table 3: Comparison of code lengths as a function of sequence length N.

Symbol probability	N	$N * H$	AC	AC with fixed length $\varepsilon = 0.03$	Proposed method with maximum $\varepsilon = 0.03$
$P(\text{a}) = 1/3$ $H =$ entropy $= .9183$	10	9.183	9.1890	9.7390	9.4470
	100	91.83	91.9100	96.3610	94.0330
	1000	918.3	918.6810	962.7090	941.2350
	10000	9183	9183.700	9622.410	9404.46
$P(\text{a}) = 5/6$ $H =$ entropy $= .5917$	10	5.917	6.0410	6.4450	6.0100
	100	59.17	65.7390	69.6020	67.3620
	1000	591.7	650.3070	694.9340	672.7320
	10000	5917	6500.250	6940.01	6717.720

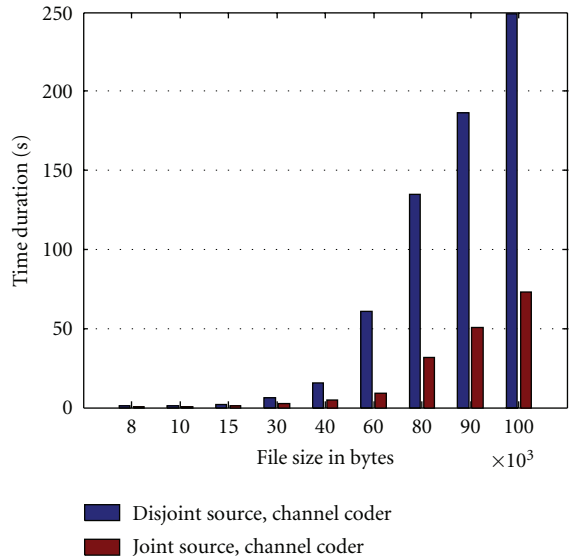

Figure 4: Comparison of AC with forbidden symbol and cascaded arithmetic coding with convolutional coder.

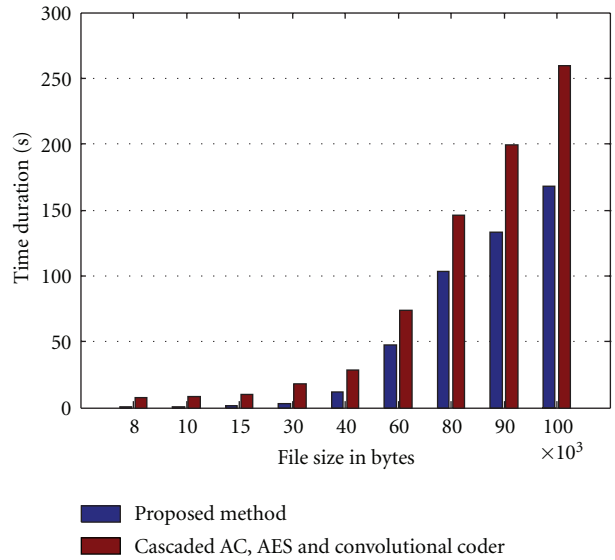

Figure 5: Comparison of proposed system and cascaded arithmetic coding with AES and convolutional coder.

increase computational complexity. This extracomputational complexity of joint AC and channel coding in comparison with the complexity of three disjoint coders is very small.

It is relevant to consider a system consisting of a traditional arithmetic encoder followed by AES, which, of course, would also deliver security and compression. Since AES was designed for efficient hardware implementation, it is extremely fast when it is fully pipelined in hardware [19]. However, because a traditional arithmetic coder needs to work sequentially, the AC cannot easily be parallelized and becomes a bottleneck in a combined AC/AES system [7]. AES consists of 40 sequential transformation steps composed of simple and basic operations such as table lookups, shifts, and XORs. For a block size of 128, these steps require a total number of 19 shifts, use of 336 bytes of memory, and the XORing of approximately (the exact requirement is data dependent) 608 bytes of data. But, our proposed technique adds a maximum of 20 bytes of memory, no XOR, and no shift operations to conventional AC. For a block size of 128 and a source with binary alphabets, our proposed method adds as much as $128 \times 2 \times 2$ operations to conventional arithmetic coding operations. This number is much smaller than the number of operations added to the AC with disjoint coders. Figure 5 compares time duration required by binary arithmetic coding for $p(\text{a}) = 1/3$ followed by AES with a block size of $N = 128$ and 1/2 feedforward convolutional encoder with our proposed method. We can see that our system takes much shorter time than a cascaded system.

Our proposed technique can be implemented utilizing techniques similar to those used in traditional arithmetic coding and can benefit from the same optimizations for speed, finite precision, and so forth. Inserting the forbidden symbol to the probability table adds no complexity to arithmetic coder; only establishing the probability table and searching the look up table increase the amount of memory needed to store the look up table and the probability of forbidden symbols. In addition, division of the forbidden symbol and updating the probability of symbols by factor $(1 - \gamma)$ in each recursion introduce an additional multiplication though, as with traditional arithmetic coding, faster

algorithms that replace the multiplications with simpler operations can be introduced [20].

4.3. Security Analysis. A good encryption procedure should be robust against all kinds of cryptanalytic, statistical, and brute-force attacks. In this section, we discuss the security analyses of the proposed encryption scheme. This includes statistical analysis, key space analysis, and sensitivity analysis of the proposed encryption scheme with respect to the key and plaintext, and so forth. to prove that the proposed cryptosystem is secure against the most common attacks.

4.3.1. Key Space. For a secure encryption algorithm, the key space should be large enough to make the brute force attack infeasible. The main private information in our proposed scheme is the key used in the PRNGs; each of them is as long as 128 bits. These PRNGs generate random sequences which are used by the proposed technique as a secret key in each recursion.

The proposed cipher has $2^{128\times 3}$ different combinations of the secret key, and key space of our proposed method is larger than that of the methods introduced in [1, 12]. A cipher with such a long key space is sufficient for reliable practical use in multimedia communications.

As mentioned above, the proposed encoder uses generated random sequences as its secret key in each recursion. In [13] there are only two possible choices in one recursion: at the beginning of the probability table or at the end. Even though the swapping probability is also used as a key parameter in this method, but there are other keys, γ_1 and γ_2, and attacker must decode received sequence using all possible seeds, S, or γ_1 and γ_2 for accessing correct data.

If precision of γ is set to 16 bits, one should try 2^{32} trails for estimating each forbidden symbol in one recursion and 2^3 trails for finding the situation of the probability table in each recursion; therefore, the actual key space in each recursion can be 2^{34} times larger than the key space in [13]. In this proposed method, if we suppose that the key S is known by the attacker, he cannot find out what random value at which positions is added, and as long as the attacker is not aware of the value of the forbidden symbols he cannot access the status of the probability table in each recursion.

4.3.2. NIST SP 800-22 Test for Cipher. In this study, NIST SP 800-22 [21] tests are used for testing the randomness of the cipher. The NIST Test Suite is a statistical package consisting of 16 tests that were developed to test the randomness of binary sequences, with arbitrary lengths, produced by either hardware or software-based cryptographic random or Pseudorandom Number Generators. These tests focus on a variety of different types of nonrandomness that could exist in a sequence. Hence, in this test the cipher sequence, whose length is 10^6, is examined. The results of testing the randomness of the cipher are shown in Table 4. We can conclude from Table 4 that the cipher which was encrypted from this encoder is stochastic and it has robustness against known cipher-text attack.

4.3.3. Sensitivity Analysis. An ideal procedure of data encryption should be sensitive to both the secret key and the plaintext. The change of a single bit in either the secret key or the plaintext should produce a completely different encrypted data. To prove the robustness of the proposed scheme, we performed sensitivity analysis with respect to both the secret key and the plaintext.

(A) Sensitivity Analysis of the Cipher to Key. For testing the key sensitivity of the proposed coder, we performed the following steps:

(a) changing one bit of *S1* which determined the forbidden symbol length in each recursion,

(b) changing one bit of *S2* which divided the forbidden symbol into two different forbidden symbols in each recursion,

(c) changing one bit of *S3* which determined the probability table in each recursion,

(d) changing just one bit of the three main keys.

It is not easy to compare the encrypted outputs by simply observing them. Thus, for the comparison, we calculated the correlation between the corresponding bits of the four encrypted data by (4) [22]:

$$C_r = \frac{N\sum_{j=1}^{N}\left(x_j \times y_j\right) - \sum_{j=1}^{N} x_j \times \sum_{j=1}^{N} y_j}{\sqrt{\left(N\sum_{j=1}^{N} x_j^2 - \left(\sum_{j=1}^{N} x_j\right)^2\right)\times\left(N\sum_{j=1}^{N} y_j^2 - \left(\sum_{j=1}^{N} y_j\right)^2\right)}}, \tag{4}$$

where, x_j and y_j are the values of corresponding bits in the two encrypted outputs to be compared and N is the total number of output bits.

We performed the above mentioned steps for several different keys. Then, we calculated the correlation coefficient for the encoded sequences by using (4). In all the cases, very small correlation coefficients of the corresponding outputs were obtained. For instance, Table 5 shows the correlation coefficients between encoded sequences with *S1*, *S2,* and *S3* keys for the outputs from the steps (a) to (d) based on changing the first bits of the keys.

As the Table 5 shows, no correlation exists among the three encrypted outputs even though these have been produced by using only slightly different secret keys. Also, based on the comparison of outputs of the proposed scheme for a large number of inputs, it was found that changing one symbol in the plaintext will result in a completely different output by more than 99%. This shows that different inputs even in one symbol will result in different outputs.

It can be also concluded from this table that all the ambiguities of the proposed coder are independent from each other. Therefore, even if the attacker finds access to one of the keys, no information about the other keys is released by that one.

Table 4: Sp 800-22 tests results of cipher.

Statistical test		P-value	Results
Monobit		1.1	success
Block frequency ($m = 128$)		.2251	success
runs		.8513	success
Rank		.2335	success
Spectral DFT		.8513	success
Nonoverlapping templates ($M = 1032$, $B = 110101010$)		.1799	success
Overlapping templates ($m = 9$, $M = 933$, $B = 110101010$)		1	success
Serial	P-value 1	.9640	Success
	P-value 2	.8729	success
Cumulative sums	Forward	.7573	Success
	reverse	.6686	success
Random excursions (state x)	$X = -4$	.9343	Success
	$X = -3$	.8757	Success
	$X = -2$	.9008	Success
	$X = -1$	.8787	Success
	$X = 1$	.2435	Success
	$X = 2$	.8922	Success
	$X = 3$	.6260	Success
	$X = 4$	.8816	success
Random excursions variant (state x)	$X = -9$	.8319	Success
	$X = -8$	.9067	Success
	$X = -7$	.8503	Success
	$X = -6$	.9922	Success
	$X = -5$	.6578	Success
	$X = -4$	.6592	Success
	$X = -3$	.9653	Success
	$X = -2$	.7504	Success
	$X = -1$	.6973	Success
	$X = 1$	.6269	success
	$X = 2$	.7790	success
	$X = 3$	.4957	success
	$X = 4$	.4850	success
	$X = 5$	.8543	success
	$X = 6$	.2173	success
	$X = 7$	.6084	success
	$X = 8$	.5193	success
	$X = 9$	.5294	success

(B) Sensitivity Analysis of Cipher to Plaintext. Generally, attacker may make a slight change in the plaintext. In order to test the influence of changing a single bit in the original data, the correlation coefficients between the corresponding output sequences were calculated for the changes in the input sequence. As expected, the correlation coefficients were very small.

Table 5: Correlation coefficients of different outputs.

Output	Correlation coefficient
Changing first bit of *S1*	0.0228
Changing first bit of *S2*	0.0170
Changing first bit of *S3*	−0.0045
Changing first bit of *S1, S2, S3*	−0.0350

Since the proposed coder is simulated for binary inputs and the output is also binary, we can calculate the changing bit rates of the cipher instead of correlation coefficients. Change of one bit in the plaintext should make theoretically a 50% difference [22] in the bits of the cipher. We also developed a test for the changing rate of the cipher bits. The changing rate was 49.41%. For all these reasons, the proposed scheme of this study proves to be sensitive to the changes in the input, hence, an ideal coder.

4.3.4. Different Attacks. According to both the above analyses and the following reasons, the proposed algorithm is resistant to the chosen plaintext attacks.

(i) The model dynamically reorders the frequency of the input symbols according to the length of random forbidden symbols in each recursion.

(ii) The output from the engine is in the form of words with variable sizes so the individual bits of the output corresponding to the inserted symbols could not be determined.

The entropy, $H(S)$, of a message source, S, can be calculated by (5)

$$H(S) = \sum_{s_i} P(s_i) \log_2 \frac{1}{P(s_i)} \text{ bits}, \tag{5}$$

where $P(s_i)$ represents the probability of symbol s_i. The entropy is expressed in bits. If the source emits 2 symbols with equal probability, that is, $S = \{s_1, s_2\}$, then the entropy is $H(S) = 1$, corresponding to a true random sequence. The system test real entropy value is 0.9974. So the system can resist the entropy attacks.

Another large class of attacks is based on the analysis of statistical properties of the output bit stream $B = B_1 B_2 \cdots B_{N_c}$, where N_c is the output length. It is thus important to investigate the statistics of B. Various simulations showed that the output of the proposed coder had $P(B_i = 0) = P(B_i = 1) = 1/2$, for any i. Therefore, from the first-order statistics, the attacker cannot find any information regarding the secret key.

Alternatively, the attacker may wish to recover the key stream which is used in the proposed method. Suppose that the input symbol sequence length is N. The length of the key stream used in the method is then $L_N = (2 \times 16 \times N) + 3 \times N$. Assume that the generated bit stream is of length N_c. Then, the total complexity of breaking the key stream is $2^{N_c + L_N}$. In the case that the input symbol sequence length is sufficiently

large which makes $2^{N_c+L_N} > 2^{128\times 3}$, the attacker would rather use the brute-force attack to break the secret key utilized in the PRNGs.

A pseudorandom sequence is vulnerable to the known plaintext attacks; since there is a given known input sequence, the attacker can compare the joint source-channel coder and the proposed coded sequences and attempt to find the added subintervals and their locations. To increase the security, an efficient key distribution protocol could be also explored in our algorithm to provide a sufficient encryption.

5. Conclusion

In this paper, a scheme has been presented which combines compression, error detection, and data encryption. The proposed technique by adding a little complexity to CED provides security. It adds two random subinterval μ_1 and μ_2 to the probability interval in each iterative coding step and controls the locations of the forbidden symbol by a PRNG with a seed, S, while the key is $\mathbf{K} = (S, \gamma_1, \gamma_2)$ in each recursion. Moreover, it easily switches to standard arithmetic coding by setting γ_1 and γ_2 equal to zero when the data do not need to be protected. This coder causes the added redundancy to be almost halved without any special effect on error detection capability. The proposed technique is less complicated and faster than cascaded systems; therefore, they are more suitable for real-time applications. The technique can be also extended to selectively encrypting data and images. This proposed method can be used in ARQ systems for error detection and error correction.

Acknowledgment

The authors would like to thank ITRC (Iran Telecommunication Research Center) for the invaluable assistance and funding for this paper.

References

[1] E. Magli, M. Grangetto, and G. Olmo, "Joint source, channel coding, and secrecy," *EURASIP Journal on Information Security*, vol. 2007, Article ID 79048, 7 pages, 2007.

[2] H. Kaneko and E. Fujiwara, "Joint source-cryptographic-channel coding based on linear block codes," in *Applicable Algebra in Engineering, Communication and Computing*, vol. 4851 of *Lecture Notes in Computer Science*, pp. 158–167, 2007.

[3] R. Bose and S. Pathak, "A novel compression and encryption scheme using variable model arithmetic coding and coupled chaotic system," *IEEE Transactions on Circuits and Systems*, vol. 53, no. 4, pp. 848–857, 2006.

[4] D. Xie and C.-C. J. Kuo, "Multimedia encryption with joint randomized entropy coding and rotation in partitioned bitstream," *EURASIP Journal on Information Security*, vol. 2007, Article ID 35262, 12 pages, 2007.

[5] A. Moffat, R. M. Neal, and I. H. Witten, "Arithmetic coding revisited," *ACM Transactions on Information Systems*, vol. 16, no. 3, pp. 256–294, 1998.

[6] T. Cover and J. Thomas, *Elements of Information Theory*, John Wiley & Sons, New York, NY, USA, 1991.

[7] H. Kim, J. Wen, and J. D. Villasenor, "Secure arithmetic coding," *IEEE Transactions on Signal Processing*, vol. 55, no. 5, pp. 2263–2272, 2007.

[8] C. Boyd, J. G. Cleary, S. A. Irvine, I. Rinsma-Melchert, and I. H. Witten, "Integrating error detection into arithmetic coding," *IEEE Transactions on Communications*, vol. 45, no. 1, pp. 1–3, 1997.

[9] J. Sayir, *On Coding By Probability Transformation*, Hartung-Gorre, Konstanz, Germany, 1999.

[10] J. Sayir, "Arithmetic coding for noisy channels," in *Proceedings of the Information Theory and Communication Workshop*, pp. 69–71, IEEE, 1999.

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

[12] M. Grangetto, E. Magli, and G. Olmo, "Multimedia selective encryption by means of randomized arithmetic coding," *IEEE Transactions on Multimedia*, vol. 8, no. 5, Article ID 1703505, pp. 905–917, 2006.

[13] P. Teekaput and S. Chokchaitam, "Secure embedded error detection arithmetic coding," in *Proceedings of the 3rd International Conference on Information Technology and Applications (ICITA '05)*, pp. 568–571, IEEE, July 2005.

[14] R. Anand, K. Ramchandran, and I. V. Kozintsev, "Continuous error detection (CED) for reliable communication," *IEEE Transactions on Communications*, vol. 49, no. 9, pp. 1540–1549, 2001.

[15] J. M. Wozencraft, and B. Reiffen, *Sequential Decoding*, MIT Press, Cambridge, Mass, USA, 1961.

[16] R. M. Fano, "A heuristic discussion of probabilistic decoding," *IEEE Transactions Information Theory*, pp. 64–74, 1963.

[17] B. D. Pettijohn, K. Sayood, and M. W. Hoffman, "Joint source/channel coding using arithmetic codes," in *Proceedings of the Data Compression Conference (DDC '00)*, pp. 73–82, Snowbird, Utah, USA, March 2000.

[18] B. D. Pettijohn, M. W. Hoffman, and K. Sayood, "Joint source/channel coding using arithmetic codes," *IEEE Transactions on Communications*, vol. 49, no. 5, pp. 826–835, 2001.

[19] A. Hodjat and I. Verbauwhede, "Area-throughput trade-offs for fully pipelined 30 to 70 Gbits/s AES processors," *IEEE Transactions on Computers*, vol. 55, no. 4, pp. 366–372, 2006.

[20] M. Grangetto, E. Magli, and G. Olmo, "Multimedia selective encryption by means of randomized arithmetic coding," *IEEE Transactions on Multimedia*, vol. 8, no. 5, pp. 905–917, 2006.

[21] A. Rukhin, J. Soto, J. Nechvatal et al., "A statistical test suite for random and pseudorandom number generators for cryptographic applications," NIST Special Publication 800-22, May 2001.

[22] X. Tong, M. Cui, and Z. Wang, "A new feedback image encryption scheme based on perturbation with dynamical compound chaotic sequence cipher generator," *Optics Communications*, vol. 282, no. 14, pp. 2722–2728, 2009.