A Secure Arithmetic Coding Algorithm Based on Integer Implementation

Yuh-Ming Huang
Dept. of Computer Science and Information Engineering
National Chi Nan University
Puli, Nantou, Taiwan, ROC
ymhuang@csie.ncnu.edu.tw

Yin-Chen Liang
Dept. of Computer Science and Information Engineering
National Chi Nan University
Puli, Nantou, Taiwan, ROC
s94321908@ncnu.edu.tw

***Abstract*—This paper presents a novel modification of integer arithmetic code, which performs compression and encryption at the same time. It is quite different from the randomized arithmetic code (RAC), the interval splitting arithmetic code (ISAC), and the secure arithmetic code (SAC). In security, we apply the Pseudo-Random Bit Generator (PRBG) and the Secure Hash Algorithm (SHA-256) to construct the *key vector*. Each bit of the *key vector* known to both the encoder and decoder is used to determine whether the *source intervals* respectively allocated to each of the different symbols are needed to be adjusted prior to encoding each new symbol. Experimental results indicate that our proposed code does not compromise the coding efficiency.**

Keywords-arithmetic coding; randomized arithmetic coding; secure arithmetic code; encryption; data compression; pseudorandom bit generator ; Secure Hash Algorithm

I. INTRODUCTION

The demand in joint compression and encryption is growing more conspicuous with the development in multimedia applications, such as digital rights management [1].

There are three approaches to achieve both compression and encryption. First, the simplest approach is to use a traditional entropy code for compressing, and then the compressed multimedia data is totally encrypted by a traditional cryptographic algorithm such as the Data Encryption Standard (DES) [2] or the Advanced Encryption Standard (AES) [3]. This approach is poorly suited to real-time multimedia applications since its computational complexity is extremely high. Second, the computational complexity can be reduced significantly by the selective encryption approaches [4-5] since they only encrypt the important portion of the multimedia data. Third, the entropy code, such as the Huffman code or the arithmetic code, is modified to possess the functionality of encryption. In this paper, we focus on the third approach.

In Huffman coding, Wu and Kuo [6-8] proposed a multiple Huffman table (MHT) scheme that alternately uses the multiple statistical models in a secret order. The advantage of this kind of scheme is to offer a certain level of security and an uninfluenced coding efficiency simultaneously. Unfortunately, MHT method is just secure against ciphertext-only attack, and is vulnerable to know-plaintext attack and chosen- plaintext attack [9].

In arithmetic coding (AC) [11-12], Grangetto *et al*. [13] presented a RAC based on randomly swapping of two intervals during the process of binary AC and it was utilized to encrypt the JPEG2000 coded images [10]. According to a secret key, Kim *et al.* [14-15] presented a binary ISAC to offer the security by splitting the interval of each symbol. Jakimoski *et al.* [16] analyzed that the key-based interval splitting AC is vulnerable to known-plaintext attack and pointed out that the randomized arithmetic coding is not as secure as the authors claimed. Kim *et al.* [17] provided a SAC which applies two permutations to the input symbol sequence and the output symbol sequence for increasing the security. However, Zhou *et al.* [18-20] and Sun *et al.* [21] independently proved that it is still not secure under the chosen-plaintext attack and known-plaintext attack. Unfortunately, the splitting AC will compromise the coding efficiency. It was shown in the Table 1 of [15] or in the Table VI of [17]. This motivates us to design a new secure arithmetic code without influencing the coding efficiency.

In this paper, we provide a novel scheme to adjust the sizes of intervals associated with the chosen symbols, which are determined by a secret key. Our proposed technique can offer a high degree of security and does not compromise the coding efficiency. The paper is organized as follows. Section II briefly introduces the integer AC. Our proposed scheme is presented in Section III. The evaluation of the security and the coding efficiency is discussed in Section IV. Finally, some concluding remarks are made in Section V.

II. ARITHMETIC CODING BASED ON INTEGER IMPLEMENTATION

For a source with a sequence of symbols $(x_1, x_2, ..., x_N)$, where $x_n \in A$, $1 \le n \le N$ and $\mathcal{A}$ is the set of ASCII symbols. The principle of AC is to recursively partition a *source interval* $[L, U)$ into non-overlapped subintervals according to the source symbol probabilities. The initial *source interval* is set to the unit interval [0, 1) of real numbers. Once a new symbol is encoded, one of those subintervals is selected as the new *source interval* according to the value of this new symbol. If this new symbol is the last symbol, the new *source interval* is the *code interval*, which is the interval a tag (any real number in the interval) will reside. The binary representation in finite length of this tag is the arithmetic code of the symbol sequence. For sources with highly skewed source symbol probabilities and for sources with longer sequence length,

subintervals may become too small to be accurately handled by a finite-precision computer. The problem is solved by the technique, called incremental encoding [22]. In addition, by using this technique, the binary bits associated with the tag can be generated sequentially without waiting until the entire sequence has been observed.

Instead of float-point implementation to AC, integer implementation is widely adopted since it can decrease the computational cost of AC without inducing significant compression loss. The resulting code is called integer arithmetic code. The initial *source interval* $[L^{(0)}, U^{(0)}]$ is equal to [0, W-1] of integers, where $W = 2^p$, p is the size of arithmetic code corresponding to a single symbol. Similar to float-point AC, the *source interval* $[L^{(n-1)}, U^{(n-1)}]$ is divided and the new *source interval* $[L^{(n)}, U^{(n)}]$ is updated as follows:

$$\begin{cases} L^{(n)} = L^{(n-1)} + \lfloor (U^{(n-1)} - L^{(n-1)} + 1) F_X(x_n - 1) \rfloor \\ U^{(n)} = L^{(n-1)} + \lfloor (U^{(n-1)} - L^{(n-1)} + 1) F_X(x_n) \rfloor - 1, \end{cases} \quad (1)$$

which is determined by the current symbol x_n to be encoded, where x_n denotes the nth observed symbol, x_n-1 denotes the symbol preceding x_n in the alphabet A, $L^{(n-1)}$ and $L^{(n)}$ respectively denotes lower limit of (n-1)-th and n-th *source interval*, $U^{(n-1)}$ and $U^{(n)}$ respectively denotes upper limit of (n-1)-th and n-th *source interval*, and the $F_X(x_n)$ is the cumulative probability obtained by summing up the probability of 1st symbol to that of x_n in the alphabet $\mathcal{A}$. There is one major problem that the values of $L^{(n-1)}$ and $U^{(n-1)}$ become closer and closer together as n gets larger and subintervals obtained by the equation (1) will not be non-overlapped. To avoid this situation, renormalization is required by doubling the size of the *source interval* through the following three mappings.

- E_1: If the current source interval [L, U] lies entirely in the lower half of [0, W-1], i.e. [0, W/2-1], then emitting a bit 0 and *Scale3* bits 1, and linearly expanding [L, U] to [2L, 2U+1]. *Scale3* is reset to 0, where the variable *Scale3* is used to count the number of E_3 mappings.
- E_2: If the current source interval [L, U] lies entirely in the upper half of [0, W-1], i.e. [W/2, W-1], then emitting a bit 1 and *Scale3* bits 0, and linearly expanding [L, U] to [2L-W, 2U-W +1]. *Scale3* is reset to 0.
- E_3: If the current source interval [L, U] lies entirely in the interval [W/4, 3W/4-1], then linearly expanding [L, U] to [2L-W/2, 2U-W/2 +1] and increasing the value of *Scale3* by 1.

III. The Proposed Secure Arithmetic Coding Algorithm

In this section, we design a modified integer arithmetic code which owns the capabilities of compression and encryption simultaneously. We first point out the motivation of our design and then introduce the secure arithmetic code in details.

A. Motivation

Suppose we have a source that generates a symbol from the alphabet $\mathcal{A}$ = {A, B, C} with the probability model p(A) = 0.5, p(B) = 0.3, and p(C) = 0.2. The entropy for this source is

$$E = -0.5\log_2 0.5 - 0.3\log_2 0.3 - 0.2\log_2 0.2 = 1.485475. \quad (2)$$

For integer AC with p = 4. By using equation (1), where x_1 = A, B, or C, and if x_n = C then x_n -1 denotes B; $F_X(A-1) = 0, F_X(A) = 0.5$, $F_X(B) = 0.8$, and $F_X(C) = 1$. The initial *source interval* [0, 15] will be partitioned into three subintervals [0, 7], [8, 11], and [12, 15], which are the *source interval*s respectively associated with the symbols A, B, and C. Hence, the actual entropy is equal to

$$E_1 = -0.5\log_2 (8/16) - 0.3\log_2 (4/16) - 0.2\log_2 (4/16) = 1.5. \quad (3)$$

Equation (3) means that the compression loss (or the redundancy) caused by integer AC is equal to E_1-E bits per symbol.

Now, if we increase the size of the *source interval* associated with the symbol with largest probability by one and decrease the size of the *source interval* associated with the symbol with smallest probability by one, then the actual entropy is equal to

$$E_2 = -0.5\log_2 (9/16) - 0.3\log_2 (4/16) - 0.2\log_2 (3/16)$$
$$= 1.498045, \quad (4)$$

which is less than E_1.

Equation (4) means that the alternative partition can have compression gain while compared to the normal partition in integer AC.

Based on this idea, the diagram of the proposed secure AC algorithm is shown in Fig. 1.

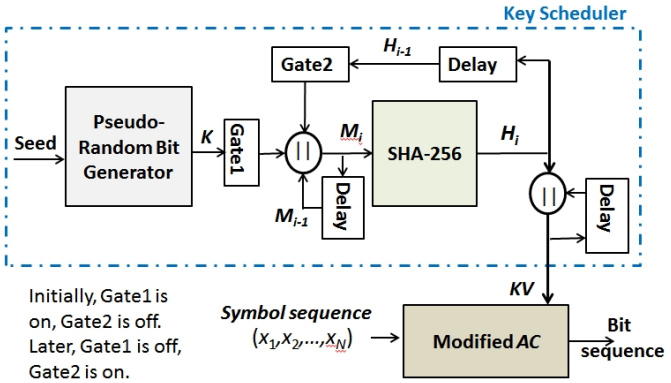

Figure 1. The diagram of our proposed scheme.

B. Key Scheduler

The operation of the key scheduler is described as follows:

Step 1) The Pseudo-Random Bit Generator (PRBG) [23] outputs a pseudo-random bit sequence $\boldsymbol{K}$, which is the secret key of our system.

Step 2) Feed the Secure Hash Algorithm (SHA-256) module with the secret key $\boldsymbol{K}$ and obtain the 1st bit sequence of length 256 denoted as H_1.

Step 3) The output bit sequence H_i and the input message M_i of SHA-256 module are concatenated as the next input message. That is, $M_i = \boldsymbol{K}||H_1||H_2||\ldots||H_{i-1}$ and

H_i = SHA-256(M_i), where $1 \le i \le T$ *and* H_0 denotes an empty bit string.

Step 4) Repeat *Step 3* for T times until the size of the bit sequence $H_1||H_2||\ldots||H_T$ (denoted as ***KV***) is not less than N. ***KV*** is used as the key vector for the secure AC.

C. Encooder

Initially, let *source interval* be equal to [0, W-1].
For i = 1 : N-1
{
  Divide the *source interval* as the traditional integer AC does.
  If ***KV***[i] = 1, Then
  {
    Increase the size of the *source interval* associated with the symbol with the largest probability by one.
    Choose the *source interval* which satisfies the following two conditions: 1) its size is greater than 1, 2) the probability of the corresponding symbol is smallest, then decrease the size by one.
  }
  // ***KV***[i] denotes the i-th bit of the key vector ***KV***.
  Encode the observed symbol x_i.
  Update the *source interval* according to the current symbol encoded.
}

TABLE I. EIGHT DIFFERENT ADJUSTMENT POLICIES USED IN AC

Key bits	Description
000	The size of each of the *source interval*s is unchanged
001	The size of the *source interval* associated with the symbol with the highest occurring frequency is increased by one
011	The size of the *source interval* associated with the symbol with the second highest occurring frequency is increased by one
010	The size of the *source interval* associated with the symbol with the third highest occurring frequency is increased by one
110	The size of each of the *source interval*s respectively associated with the symbols with the highest and the second highest occurring frequency is increased by one
111	The size of each of the *source interval*s respectively associated with the symbol with the second highest and the third highest occurring frequency is increased by one
101	The size of each of the *source interval*s respectively associated with the symbols with the highest and the third highest occurring frequency is increased by one
100	The size of each of the *source interval*s respectively associated with the symbols with the top three highest occurring frequencies is increased by one

The proposed encoding algorithm can be easily extended to the version that the size of each of two or more *source interval*s is increased by one. Table I shows that three key bits are required at a time to determine the adjustment policy for each of the *source interval*s once a symbol to be encoded. Fig. 2 gives two examples. In Fig. 2 (a), the size of the interval associated with the symbol A is increased by one, whereas the size of the interval associated with the symbol EOF is decreased by one. Other interval size is unchanged. The positions of all of the intervals are altered slightly. In Fig. 2 (b), the size of each of the intervals respectively associated with the symbol A and the symbol C is increased by one, whereas the size of the interval associated with the symbol EOF is decreased by two. Other interval size is unchanged. The positions of all of the intervals are altered slightly.

D. Decoder

In the decoder, once the secret key ***K*** is known, the key vector ***KV*** can be generated successfully. Then, the symbol sequence can be decoded correctly just by mimicking the encoder algorithm.

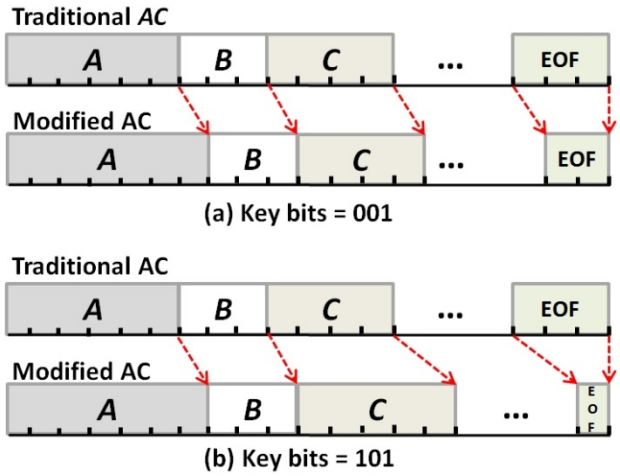

Figure 2. Alternative ways to adjust the size of each of the *source intervals*.

IV. EVALUATIONS

In this section, we first show the experimental results respectively obtained by the traditional integer AC and the modified integer AC, then discuss the security of our proposed scheme under the three known attacks.

A. Coding Efficiency

The proposed algorithm has been tested on various source benchmarks taken from the Canterbury Corpus file set (available at http://corpus.canterbury.ac.nz). These benchmarks were designed specifically for testing new compression algorithms.

Table II shows that our proposed scheme does not compromise the coding efficiency. In worst case, the compression ratio of our proposed scheme is equal to that of the traditional one.

B. Cryptanalysis

We evaluate the security of our proposed scheme under the ciphertext-only attack, the known-plaintext attack, and the chosen-plaintext attack.

1) Ciphertext-only attack

The ciphertext-only attack is the most common attack. An attacker can break the encryption scheme according to the encrypted bit stream. Assume the length of the input symbol sequence is equal to N. The complexity of breaking the key

space is 2^{128N}. Apparently, in this situation, if the input symbol sequence is sufficiently long, it is hard for the attacker to break the correct key.

2) Known-plaintext attack

In this attack model, the attacker tries to find the secret information such as the secret keys from the plaintext and its corresponding ciphertext. In order to avoid that the whole input symbol sequence is encrypted by repeatedly using one short key vector, we apply the SHA-256 hashing algorithm to generate the key vector sequence $H_1, H_2,\ldots,H_T$, all of them are different.

3) Chosen-plaintext attack

In the chosen-plaintext attack, the attacker feeds several selected symbol sequences to the encoder, and obtains the corresponding ciphertexts. However, it is still difficult for the attacker to correctly find out those chosen symbols of which the corresponding *source interval*s are adjusted in our proposed scheme. The failure of this attack is due to the fact that the chosen policy for the *source interval*s every time is not static.

TABLE II. CODING EFFICIENCY

File Name	File Size (Bytes)		
	Original File	***Traditional AC***	***Modified AC***
alice29.txt	152089	86852	86850
asyoulik.txt	125179	75242	75241
cp.html	24603	16087	16087
fields.c	11150	6984	6984
grammar.lsp	3721	2159	2159
kennedy.xls	1029744	460040	460029
lcet10.txt	426754	249097	249093
out	777	299	299
plrabn12.txt	481861	273002	273000
ptt5	513216	77965	77964
sum	38240	25481	25481
xargs.1	4227	2593	2593

V. CONCLUSIONS

Through the adjustment of the size of *source interval*, a secure integer arithmetic code is presented. The proposed code owns the capabilities of compression and encryption simultaneously. In addition, the coding efficiency is not affected. In the future, we will study on the key scheduler to further enhance the security.

ACKNOWLEDGMENT

This work was supported in part by the National Science Council, Taiwan R. O. C. , under the Contract NSC 99-2628-E-260-010.

REFERENCES

[1] D. Kundur and K. Karthik, "Video fingerprinting and encryption principles for digital rights management," *Proc. IEEE*, vol. 92, no. 6, pp.918–932, Jun. 2004.

[2] *Data Encryption Standard*, FIPS PUBS 46-2, 1993.

[3] *Advanced Encryption Standard*, FIPS PUBS 197, 2001.

[4] T. Lookabaough and D. C. Sicker, "Selective encryption for consumer applications," *IEEE Commun. Mag.*, pp. 124–129, May 2004.

[5] H. Cheng and X. Li, "Partial encryption of compressed images and videos," *IEEE Trans. Image Process.*, vol. 48, no. 8, pp. 2439–2451, Aug. 2000.

[6] C. Wu and C.-C. J. Kuo, "Efficient multimedia encryption via entropy codec design", *SPIE international symposium on electronic imaging*, San Jose, CA, Jan. 2001.

[7] D. Xie and C.-C. J. Kuo, "Enhanced multiple Huffman table (MHT) encryption scheme using key hoping," *in Proc. ISCAS*, vol. 5, pp. 568–571, May 2004.

[8] 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.

[9] J. T. Zhou, Z. Q. Liang, Y. Chen, and O. C. Au, "Security analysis of multimedia encryption schemes based on multiple Huffman table," *IEEE Signal Process. Lett.*, vol. 14, pp. 201–204, Mar. 2007.

[10] M. Grangetto, A. Grosso, and E. Magli, "Selective Encryption of JPEG 2000 Images by Means of Randomized Arithmetic Coding," *Multimedia Signal Processing, 2004 IEEE 6th Workshop on*, pp. 347–350, 29 Sept.-1 Oct. 2004.

[11] H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic Coding for Data Compression," Communication of the ACM, Vol. 30, Jun. 1987.

[12] P. G. Howard and J. S. Vitter, "Arithmetic coding for data compression," *Proc. of IEEE*, vol. 82, pp.857 - 865, 1994.

[13] M. Grangetto, E. Magli, and G. Olmo, "Multimedia Selective Encryption by Means of Randomized Arithmetic Coding," *IEEE Trans. Multimedia*, vol. 8, no. 5, pp. 905–917, Oct. 2006.

[14] H. Kim, J. Villasenor, and J. Wen, "Secure Arithmetic Coding Using Interval Splitting," *Signals, Systems and Computers, 2005. Conference Record of the Thirty-Ninth Asilomar Conference on*, pp. 1218–1221, Oct. 28 – Nov. 1, 2005.

[15] J. T. Wen, H. Kim, and J. D. Villasenor, "Binary arithmetic coding with key-based interval splitting," *IEEE Signal Proc. Letters*, vol. 13, pp. 69–72, Feb. 2006.

[16] G. Jakimoski and K. P. Subbalakshmi, "Cryptanalysis of some multimedia encryption schemes," *IEEE Trans. Multimedia*, vol. 10 no. 3, pp. 330–338, Apr. 2008.

[17] H. Kim, J. T. Wen, and J. D. Villasenor, "Secure arithmetic coding," *IEEE Trans. Signal Proc*., vol. 55, pp. 2263–2272, May 2007.

[18] J. Zhou, O. C. Au, X. Fan, and P. H. Wong, "Joint security and performance enhancement for secure arithmetic coding," *In Proceedings of the IEEE International Conference on Image Processing (ICIP)*, pp. 3120-3123, 2008.

[19] J. Zhou, O. C. Au, P. H. Wong, and X. Fan. "Cryptanalysis of secure arithmetic coding," *In Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 1769-1772, Apl. 2008.

[20] J. Zhou, O. C. Au, and P. H. W. Wong, "Adaptive chosen-ciphertext attack on secure arithmetic coding," *IEEE Transactions on Signal Processing*, vol. 57, no. 5, pp. 1825–1838, 2009.

[21] H. M. Sun, K. H. Wang, and W. C. Ting, "On the Security of Secure Arithmetic Code," *IEEE Trans. Information Forensics and Security*, vol.4, no.4, pp.781-789, Dec. 2009.

[22] K. Sayood, *Introduction to Data Compression*, 3ed, Morgan Kaufmann Publishers, 2005.

[23] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1996.