CRYPTANALYSIS OF SECURE ARITHMETIC CODING

Jiantao Zhou, Oscar C. Au, Peter H. Wong and Xiaopeng Fan

Department of Electrical and Computer Engineering
Hong Kong University of Science and Technology
Clear Water Bay, Hong Kong, P. R. China.

ABSTRACT

This work investigates the security issues of the recently proposed secure arithmetic coding (AC), which is an encryption scheme incorporating the interval splitting AC with a series of symbol and codeword permutations. We propose a chosen-ciphertext attack which is capable of recovering the key vectors for codeword permutations with complexity $\mathcal{O}(N)$, where N is the symbol sequence length. After getting the key vectors for codeword permutations, we can remove the codeword permutation module, and the resulting system has already been shown to be insecure in the original paper [5].

Index Terms— Cryptanalysis, Arithmetic Coding, Digital rights management.

1. INTRODUCTION

The problem of efficient multimedia data encryption has recently gained much attention in both academic and industrial fields [1, 2, 3, 4, 5]. A straightforward approach to achieve multimedia security is to encrypt the entire data by using a traditional cryptographic algorithm, such as DES and AES. However, this kind of method is not computationally efficient, and many types of advanced multimedia processing cannot be applied directly in the encrypted bit stream [1, 2, 5].

The recent trend in multimedia encryption has placed more attention on integrating encryption and compression by introducing randomness into the entropy coder, e.g., Huffman coder and arithmetic coder [1, 3, 4, 5]. In [4], Wen *et al.* modified the traditional AC by removing the constraint that a single continuous interval is used for each symbol, while preserving the sum of the lengths of intervals allocated to each symbol [4]. The modified AC is called interval splitting AC, and it was shown that it can provide certain level of security with vanishing coding efficiency penalty [4]. Aiming to further increase the security while not influencing the coding efficiency of the interval splitting AC, Kim *et al.* proposed the secure AC by applying a series of permutations at the symbol sequence and the codeword of the interval splitting AC. Security analysis results using cipher-only attack, known-plaintext attack and chosen-plaintext attack show that the secure AC offers very high security [5].

In this paper, we address the security issues of the secure AC. Different from the security analysis of the original paper which is from the angle of the encoder, we investigate this problem from the perspective of the decoder. This is feasible since the secure AC is essentially a symmetric cryptosystem. We present a chosen-ciphertext attack that can successfully recover the key vectors for codeword permutations with complexity $\mathcal{O}(N)$, where N is the symbol sequence length. After getting these key vectors, we can remove the codeword permutation module, and the resulting system has already been shown to be insecure in the original paper [5].

The rest of this paper is organized as follows. Section 2 briefly introduces the interval splitting AC and the secure AC. Section 3 shows the chosen-ciphertext attack for breaking the secure AC. We conclude this paper in Section 4.

Notations: Throughout this paper, we restrict our attention to the binary interval splitting AC. For a binary codeword C, we denote $(C)_d$ as its decimal representation. For example, $(011)_d = 2^{-2} + 2^{-3} = 0.375$. Let $S = s_1 s_2 \cdots s_N$ and $S' = s'_1 s'_2 \cdots s'_N$ be two binary symbol sequences of the same length N. We define the Hamming distance between them, denoted by $d_H(S, S')$, as the number of positions in which the corresponding symbols are different.

2. INTERVAL SPLITTING AC AND SECURE AC

In the traditional AC, the intervals associated with each symbol are continuous. In the interval splitting AC, however, this condition has been removed, and a more generalized constraint that the sum of the lengths of the one or more intervals associated with each symbol should be equal to its probability has been utilized [5]. This is achieved by splitting the intervals according to a sequence of splitting keys known both to the encoder and the decoder. More details about how to split the intervals and the constraints involved in the interval splitting can be found in [4].

In order to increase the security of the interval splitting AC, in [5], the secure AC was proposed, in which the major

This work was supported by Innovation and Technology Commission of the HK Special Administrative Region, China (project no. GHP/033/05). Jiantao Zhou is now visiting the University of Illinois at Urbana-Champaign, supported by the Fulbright Program.

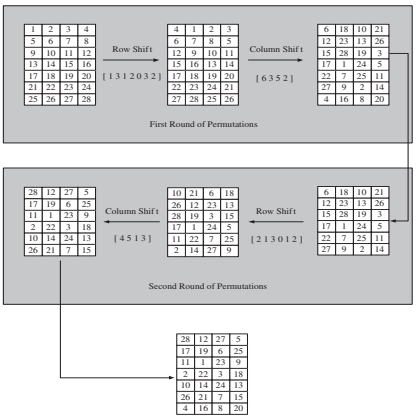

Fig. 1. Codeword permutations using the key vectors in (1), where the numbers in the blocks denote the bit indexes with left-to-right order. In other words, the codeword $c_1c_2\cdots c_{28}$ becomes $c_{28}c_{12}\cdots c_{20}$ after the codeword permutations.

difference from the interval splitting AC is that two permutation modules are applied to the symbol sequence and the codeword, respectively. The codeword permutations consist of two rounds of permutations, which begin by raster-order mapping a codeword into a block of 4 columns. From the perspective of the decoder, in the first round, the codeword is subject to two key-driven cyclic shift steps, one operating on the rows and one on the columns. Then the last four bits of the result are removed, and used to generate the key vectors used for permuting the remaining bits. After the permutations of the remaining bits, the last four bits are re-appended and the data is read out in raster-order to form the final bit stream. An example can be illustrated in Fig. 1, where the key vectors for permutations are shown below

$$\begin{aligned}\mathbf{RC}_1 &= [1\,3\,1\,2\,0\,3\,2];\ \mathbf{CC}_1 = [6\,3\,5\,2]\\ \mathbf{RC}_2 &= [2\,1\,3\,0\,1\,2];\ \mathbf{CC}_2 = [4\,5\,1\,3]\end{aligned} \tag{1}$$

Here $\mathbf{RC}_1$, $\mathbf{CC}_1$, $\mathbf{RC}_2$ and $\mathbf{CC}_2$ are used for the first round of row and column permutations, and the second round of row and column permutations, respectively.

These permutations have been shown to be efficient to resist the cipher-only attack, known-plaintext attack and chosen-plaintext attack. More details please refer to [5].

3. CRYPTANALYSIS OF SECURE AC

In the original paper [5], the stand-alone interval splitting AC and the hybrid scheme combining interval splitting AC and symbol permutations only have been shown to be insecure. Hence, in order to break the secure AC, it suffices to recover the key vectors for codeword permutations. In this section, we restrict our attention to the decoder. In the following section 3.1, we first consider a simplified case using static key vectors, where both the first round and the second round of codeword permutations are independent with the input codeword. We then in section 3.2 study the case using adaptive key vectors, where the key vectors for the second round of permutations depend on the last four bits of the result after the first round of permutations. We show that our method in section 3.1 still works subject to some appropriate modifications.

3.1. Static key vectors for codeword permutations

Before going into the details of our method, let us briefly outline the core idea of our approach by giving an example. Suppose we have already known that the 2rd, 4th, and 6th bits in a codeword of length N_c will become the last three bits after the codeword permutations. We now wish to find which bit becomes the (N_c-3)th bit after the permutations. We let $C = c_1c_2\cdots c_{N_c}$, where

$$c_i = \begin{cases} 1 & \text{for } i \in \{2,4,6\} \\ 0 & \text{otherwise.}\end{cases} \tag{2}$$

Define a set $\mathcal{A} = \{a | a \in \mathcal{Z}^+ \text{ and } 1 \le a \le N_c\}$. We then let $C'(j) = c'_1c'_2\cdots c'_{N_c}$, for $j \in \mathcal{A} - \{2,4,6\}$, where

$$c'_i = \begin{cases} 1 & \text{for } i = j \\ 0 & \text{otherwise.}\end{cases} \tag{3}$$

For example, if $N_c = 8$, then $C = 01010100$ and $C'(1) = 10000000$. After the permutations, the permuted versions of C and $C'(j)$ become $\tilde{C} = 00\cdots0111$ and $\tilde{C}'(j) = \underbrace{00\cdots01}_{j'}0\cdots0$, where j' is the location of the jth bit after permutations. If it happens that $j' = N_c-3$, then $(\tilde{C}'(j))_d-(\tilde{C})_d = (00\cdots01)_d = 2^{-N_c}$, which is the minimum distance between two binary numbers of length N_c. On the contrary, if $j' \neq N_c-3$, then $(\tilde{C}'(j))_d-(\tilde{C})_d \ge 2^{-N_c}+2^{-N_c+n}$, where $n=3$ is the number of bits whose locations after the permutations have been known. This leads to the fact that if $j' = N_c-3$, then the decoded symbol sequences of C and $C'(j)$ are very similar, even identical if N_c is sufficiently large, which can be measured by their Hamming distance. While if $j' \neq N_c-3$, then it is very likely that the decoded symbol sequences differ in many symbols. In this way, we can determine which bit becomes the (N_c-3)th bit after the permutations. Following similar fashion, we can establish the correspondence of the bit locations before and after the permutations.

For the sake of better illustration, in the following, we include an example with the following configurations.

- $p(\mathbf{A}) = 2/3$ and $p(\mathbf{B}) = 1/3$. $N = 16$.
- The key vector for interval splitting AC is

$$\mathbf{K} = \Big[\frac{5}{16}\ \frac{5}{16}\ \frac{1}{16}\ \frac{7}{16}\ \frac{11}{16}\ \frac{15}{16}\ \frac{9}{16}\ \frac{7}{16}\ \frac{5}{16}\ \frac{13}{16}\ \frac{3}{16}\ \frac{15}{16}\ \frac{9}{16}\ \frac{11}{16}\ \frac{1}{16}\ \frac{13}{16}\Big] \tag{4}$$

- The key vectors for symbol permutations

$$\mathbf{RS} \quad = \quad [1\,2\,1\,3];\ \mathbf{CS} = [0\,3\,1\,2] \tag{5}$$

  where $\mathbf{RS}$ is the vector for row permutations and $\mathbf{CS}$ is the vector for column permutations.

- The key vectors for codeword permutations are shown in (1).

Using chosen-ciphertext attack to find the key vectors for codeword permutations, we perform the following steps.

Step 1: Let $C^0 = c_1c_2\cdots c_{N_{c,\max}} = 00\cdots0$, where $N_{c,\max} = \lceil -N\log_2 p(\mathbf{B})\rceil + 2$. Input the codeword C^0 to the decoder, and obtain the decoded symbol sequence $\tilde{S}^0$.

Step 2: Let $C'(j) = c'_1c'_2\cdots c'_{N_{c,\max}} = \underbrace{00\cdots01}_{j}0\cdots0$, for $1 \le j \le N_{c,\max}$. Input the codewords $C'(j)$ to the decoder, and obtain the corresponding symbol sequences $\tilde{S}^j$.

Table 1 gives the experimental results of the codeword and symbol sequence pairs using the configurations defined above, where only the data with minimum Hamming distance from $\tilde{S}^0$ are kept. It can be seen that $\tilde{S}^i$ has zero Hamming distance from $\tilde{S}^0$, where $i \in \mathcal{I} = \{4, 7, 8, 13, 14, 15, 16, 20, 21, 24, 26\}$. We can then decide that c_i, for $i \in \mathcal{I}$, correspond to the last eleven bits after the codeword permutations, although so far we still do not know their order. It can be found that this result is consistent with Fig. 1.

Step 3: Let $C = c_1c_2\cdots c_{N_{c,\max}}$, where

$$c_i = \begin{cases} 1 & \text{for } i \in \mathcal{I} \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

Define $\mathcal{A} = \{a | a \in \mathcal{Z}^+ \text{ and } 1 \le a \le N_{c,\max}\}$. Let also $C'(j) = c'_1c'_2\cdots c'_{N_{c,\max}}$, for $j \in \mathcal{A} - \mathcal{I}$, where

$$c'_i = \begin{cases} 1 & i = j \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

Input C and $C'(j)$ to the decoder and obtain their decoded symbol sequences, denoted by $\tilde{S}$ and $\tilde{S}^j$, respectively.

Table 2 gives the experimental results of the decoded symbol sequences of C and all $C'(j)$ using aforementioned configurations, where only the data with minimum Hamming distance from $\tilde{S}$ are shown. It should be noted that in this step we only need to decode C, and all the other results can be directly obtained from ***Step 1***, although these results are not shown in Table 1 due to limited space. Since $\tilde{S}^{10}$ and $\tilde{S}^{18}$ have zero Hamming distance from $\tilde{S}$, we can decide that the 10th and 18th bits correspond to the 16th and 17th bits after the permutations, although so far we still do not know their order.

Following similar steps, we can *uniquely* recover the location relationship of the remaining bits before and after the codeword permutations. Fig. 2 gives what we have already found about the bit location relationship before and after the codeword permutations, where $X_i \in \{10, 18\}$ and $Y_i \in \{4, 7, 8, 13, 14, 15, 16, 20, 21, 24, 26\}$.

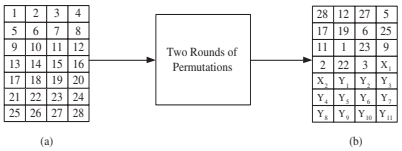

Fig. 2. The result after performing the first four steps.

To recover the key vectors for codeword permutations, a straightforward approach is to first determine the order of X_i and Y_i, based on which, derive the key vectors. However, we here propose an approach without determining the order of X_i and Y_i. Our strategy is to first reduce the two rounds of permutations partially, and based on the nature of the cyclic permutations, to recover the key vectors. In order to partially reduce the two rounds of permutations, we decrease the lengths of the codewords input to the decoder, namely, we make $N_c < N_{c,\max}$. We perform the following.

Step 4: Let $N_c = N_{c,\max} - 10$, and define C^0 and $C'(j)$ similar to ***Step 1*** and ***Step 2***, by replacing $N_{c,\max}$ with N_c. Input C^0 and $C'(j)$ to the decoder and obtain their decoded symbol sequences.

Here we choose $N_c = N_{c,\max} - 10$ is because, from the result of ***Step 1*** and ***Step 2***, we know that in this case, we can detect a unique j such that $C'(j)$ will be decoded into $\tilde{S}^0$. Notice that the last four bits of the permuted codeword after two rounds of permutations are identical to those after the first round of permutations, as can also be seen from Fig. 1. We can determine that the obtained jth bit is the last bit after the first round of permutations, based on which we can derive one column permutation parameter $CC_{1,m}$ and one row permutation parameter $RC_{1,n}$ for the first round of permutations, where $CC_{1,m}$ and $RC_{1,n}$ are the mth and the nth element of $\mathbf{CC}_1$ and $\mathbf{RC}_1$, respectively, $0 \le m \le 3$, and $0 \le n \le 6$.

Using a very similar technique as shown in ***Step 1***, ***Step 2*** and Table I, we can readily find that, in this case, the 7th bit becomes the last bit after the codeword permutations. Since the 7th bit originally locates at $(2, 3)$, we then can determine that $RC_{1,1} = 3$ and $CC_{1,1} = 3$.

Increasing N_c gradually to $N_c = N_{c,\max} - 9$, $N_c = N_{c,\max} - 8$, and $N_c = N_{c,\max} - 7$, and performing very similarly as shown in ***Step 4***, we can get $RC_{1,4} = 0$ and $CC_{1,2} = 5$; $RC_{1,2} = 1$ and $CC_{1,3} = 2$; and $CC_{1,0} = 6$.

Since now we have already known $\mathbf{CC}_1$, $RC_{1,1}$, $RC_{1,2}$, and $RC_{1,4}$, we can determine the exact locations of c_i, where $i \in \{5, 6, 7, 8, 9, 10, 11, 12, 17, 18, 19, 20\}$, after the first round of permutations. Then, based on these locations and the property of the cyclic permutations, we can recover all the remaining permutation parameters for both rounds of permutations. From example, we know that after the first round of permutations, the location of the 6th bit is (1,1). After the second

Table 1. Experimental Results of ***Step 1*** and ***Step 2***.

Input Codeword	Symbol Sequence	Hamming Distance
C^0: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	$\tilde{S}^0$: **A A B A B A A B B A B A B A B B**	—
$C'(4)$: 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	$\tilde{S}^4$: **A A B A B A A B B A B A B A B B**	0
$C'(7)$: 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	$\tilde{S}^7$: **A A B A B A A B B A B A B A B B**	0
$C'(8)$: 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	$\tilde{S}^8$: **A A B A B A A B B A B A B A B B**	0
$C'(13)$: 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	$\tilde{S}^{13}$: **A A B A B A A B B A B A B A B B**	0
$C'(14)$: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	$\tilde{S}^{14}$: **A A B A B A A B B A B A B A B B**	0
$C'(15)$: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0	$\tilde{S}^{15}$: **A A B A B A A B B A B A B A B B**	0
$C'(16)$: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0	$\tilde{S}^{16}$: **A A B A B A A B B A B A B A B B**	0
$C'(20)$: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0	$\tilde{S}^{20}$: **A A B A B A A B B A B A B A B B**	0
$C'(21)$: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0	$\tilde{S}^{21}$: **A A B A B A A B B A B A B A B B**	0
$C'(24)$: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0	$\tilde{S}^{24}$: **A A B A B A A B B A B A B A B B**	0
$C'(26)$: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0	$\tilde{S}^{26}$: **A A B A B A A B B A B A B A B B**	0

Table 2. Experimental Results of ***Step 3***.

Input Codeword	Symbol Sequence	Hamming Distance
C: 0 0 0 1 0 0 1 1 0 0 0 0 1 1 1 1 0 0 0 1 1 0 0 1 0 0 0 0	$\tilde{S}$: **A A A A B A A B B A B A B A B B**	—
$C'(10)$: 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	$\tilde{S}^{10}$: **A A A A B A A B B A B A B A B B**	0
$C'(18)$: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0	$\tilde{S}^{18}$: **A A A A B A A B B A B A B A B B**	0

round of permutations, its location is (2,3). We then can derive that $R_{2,0} = 2$ and $C_{2,2} = 1$. Due to the limited space, we omit the details here.

Now we roughly calculate the complexity of our proposed attack. In ***Step 1*** and ***Step 2***, we need to decode $N_{c,\max} + 1$ different codewords, and in each of the following steps, we only need to decode one more codeword. This results in the complexity of recovering the key vectors for codeword permutations is approximately $2N_{c,\max} = 2\lceil -N \log_2 p(\mathbf{B}) \rceil$ decoding operations, which is of the order $\mathcal{O}(N)$.

3.2. Adaptive key vectors for codeword permutations

To deal with this case, we first face the problem of determining the locations of the bits that turn out to be the last four bits after the codeword permutations. Using a very similar approach as shown in ***Step 4*** in section 3.1, we can readily find these locations. Since for different last four bits, the key vectors for the second round of permutations are different, which leads to the fact that there are $2^4 = 16$ distinct key vectors that have to be found. Suppose now we wish to find the key vectors for the case that the last four bits are equal to d_1, d_2, d_3, d_4, respectively. As we have already known the locations of bits that will eventually become the last four bits after the permutations, we can fix the bits in these locations to be d_1, d_2, d_3, d_4, respectively. After that, the key vectors become fixed. The method in section 3.1 can then be applied.

4. CONCLUSIONS

In this paper, we have addressed the security issues of the secure AC. We have proposed a chosen-ciphertext attack to recover the key vectors for codeword permutations with complexity $\mathcal{O}(N)$, where N is the symbol sequence length.

5. REFERENCES

[1] C. Wu *et al.* "Design of integrated multimedia compression and encryption systems," *IEEE Trans. Multimedia*, vol. 7, pp. 828–839, 2005.

[2] Y. Mao *et al.* "A joint signal processing and cryptographic approach to multimedia encryption," *IEEE Trans. IP*, vol. 15, pp. 2061–2075, 2006.

[3] M. Grangetto *et al.* "Multimedia selective encryption by means of randomized arithmetic coding," *IEEE Trans. Multimedia*, vol. 8, pp. 905–917, 2006.

[4] J. T. Wen *et al.* "Binary arithmetic coding with key-based interval splitting," *IEEE Signal Proc. Letters*, vol. 13, pp. 69–72, 2006.

[5] H. Kim *et al.* "Secure arithmetic coding," *IEEE Trans. SP*, vol. 55, pp. 2263–2272, 2007.