Adaptive Chosen-Ciphertext Attack on Secure Arithmetic Coding

Jiantao Zhou, *Student Member, IEEE*, Oscar C. Au, *Senior Member, IEEE*, and Peter Hon-Wah Wong, *Member, IEEE*

***Abstract*—The paper "Secure Arithmetic Coding" (in IEEE Transactions on Signal Processing, vol. 55, no. 5, pp. 2263–2272, May 2007) presented a novel encryption scheme called the secure arithmetic coding (SAC) based on the interval splitting arithmetic coding (ISAC) and a series of permutations. In the current work, we study the security of the SAC under an adaptive chosen-ciphertext attack. It is shown that the key vectors used in the codeword permutation step can be recovered with complexity $\mathcal{O}(N)$, where N is the symbol sequence length. After getting these key vectors, we can remove the codeword permutation step, and the resulting system has already been shown to be insecure in the original paper. This implies that the SAC is not suitable for the applications where the attacker can have access to the decoder. In addition, we discuss a method to jointly enhance the security and the performance of the SAC.**

***Index Terms*—Adaptive chosen-ciphertext attack, arithmetic coding, digital rights management, multimedia encryption.**

I. Introduction

The recent trend in multimedia encryption has placed more attention on integrating compression and encryption by introducing randomness into the entropy coder, e.g., Huffman coder and arithmetic coder [1]–[5]. The major advantage by using this kind of joint compression-encryption approach is that compression and encryption can be achieved in one single step, which simplifies the system design and makes it flexible for some advanced multimedia processing [1]–[6].

Wu and Kuo proposed the multiple Huffman tree (MHT) scheme that alternately uses different Huffman trees in a secret order, without influencing the coding efficiency [2]. However, we showed that this scheme is vulnerable against a chosen-plaintext attack [7]. Compared with the Huffman coding, the arithmetic coding (AC) is capable of offering higher coding efficiency, and thus becomes more and more popular in the new generation of standards, e.g., JPEG 2000 and H. 264 [8], [9]. Under this circumstance, it is natural to consider using AC to achieve encryption purposes. Along this line, Witten *et al.* suggested that an adaptive AC algorithm may provide high level of security, due to the fact that the current state in the model depends on the initial state and all of the messages encoded so far [10]. However, it was shown that the traditional implementation of AC, either using fixed model or adaptive model, cannot offer satisfactory level of security [11], [12]. In order to enhance the security, many variants of the traditional AC were proposed [1], [3]–[5], [13]–[17]. Barbir designed an encryption scheme by updating the coding probabilities in random time intervals [15]. Moo and Wu exploited the poor synchronization property of AC, and suggested a scheme by encrypting only the first few bits of the generated bit stream [13], [14]. Liu *et al.* constructed an encryption system by using random bit substitution during the encoding process of AC [16]. Ishibashi and Tanaka considered the decoding process of AC as the repetition of Bernoulli shift map, based on which, they proposed three methods to achieve security by controlling the piecewise linear map using a secret key [17]. Grangetto *et al.* described an efficient encryption scheme based on AC by randomly alternating between two coding conventions [3]. Bose and Pathak integrated a variable model arithmetic coder with a coupled chaotic system for designing an encryption scheme [5] (see also the comments from Zhou and Au [18]). 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]. This is achieved by splitting the intervals associated with one of the symbols using a key known both to the encoder and the decoder [4]. The modified AC is called the interval splitting AC (ISAC), and it was shown that it can provide certain level of security while introducing vanishing coding efficiency penalty [4]. Nevertheless, Kim *et al.* showed the insecurity of the ISAC against chosen-plaintext attacks [1]. Aiming to provide an AC system capable of offering high level of security, they suggested an enhanced version called the secure AC (SAC), by applying a series of permutations at the input symbol sequence and the output codeword of the ISAC encoder [1].

Although the SAC is originally designed for resisting chosen-plaintext attacks, it is still important to investigate the security of the SAC, as a generic cipher not targeting any specific standard, under a chosen-ciphertext attack because of two reasons: 1) a cryptographically secure cipher should be strong against all kinds of attacks including a chosen-ciphertext attack [19], and 2) in practice it is possible for the attacker to gain temporary access to the decoder and hence it is feasible to mount

Manuscript received February 28, 2008; accepted December 17, 2008. First published January 23, 2009; current version published April 15, 2009. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Christine Guillemot. This work was supported by the Innovation and Technology Commission of the Hong Kong Special Administrative Region, China (project GHP/033/05 and GHP/048/08). J. Zhou was visiting the University of Illinois at Urbana-Champaign, supported by the Fulbright program. The material in this paper was presented in part at the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Las Vegas, NV, March 30–April 4, 2008 and the IEEE International Conference on Image Processing (ICIP), San Diego, CA, October 12–15, 2008.

The authors are with the Department of Electronic and Computer Engineering, Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong, China (e-mail: eejtzhou@ust.hk; eeau@ust.hk; eepeter@ust.hk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSP.2009.2013901

a chosen-ciphertext attack. In this paper, we address the security problem of the SAC under an adaptive chosen-ciphertext attack. In other words, the attacker can have temporary access to the decoder, and the objective is to recover the keys used in the SAC. By adaptively selecting the codewords fed into the decoder and comparing their output symbol sequences, we suggest a method that can recover the key vectors used in the codeword permutation step with complexity $\mathcal{O}(N)$, where N is the symbol sequence length. After getting these key vectors, we can remove the codeword permutation step, and the resulting system has already been shown to be insecure in the original paper [1]. This indicates that the SAC is not suitable for those applications where the attacker can have access to the decoder. In addition, based on the lessons drawn from this attack, we propose a method to jointly enhance the security and the performance of the SAC.

The rest of this paper is organized as follows. Section II briefly introduces the ISAC and the SAC. Section III presents the adaptive chosen-ciphertext attack for recovering the key vectors used in the codeword permutation step. Section IV gives the method for joint security and performance enhancement of the SAC, together with the corresponding security analysis. We conclude this paper in Section V.

Notations: Throughout this paper, following the convention in [1], we restrict our attention to the binary AC, consisting of two symbols $\mathbf{A}$ and $\mathbf{B}$. Without loss of generality, we assume that $p(\mathbf{A}) \geq p(\mathbf{B})$. We denote $S(C)$ as the output symbol sequence of the SAC decoder when the input codeword is C. For a binary codeword C, we denote $(C)_d$ as its decimal representation. For example, $(011)_d = 2^{-2} + 2^{-3} = 0.375$. For a real number $0 \leq r < 1$, we denote $(r)_2^L$ as its corresponding binary codeword of length L. For example, $(0.125)_2^4 = 0010$, where we have dropped the prefix "0." without introducing ambiguity. Let $E = E_1 \cdots E_L$ be a binary bit stream. We denote $\bar{E} = \bar{E}_1 \cdots \bar{E}_L$ as its complement.

II. Review of the ISAC and the SAC

A. ISAC

The ISAC is a variant of a traditional AC in the sense that the intervals allocated to each symbol in every step of encoding may be disjoint, and the sum of their lengths is proportional to the symbol occurrence probability. Let $S = s_1 s_2 \cdots s_N$ be the symbol sequence to be encoded, and the splitting key vector be $\mathbf{K} = (k_1, k_2, \cdots, k_N)$. The encoding procedure is shown as follows.

Step 1) Set the initial interval $I = [0, 1)$, and set $i = 1$.
Step 2) Fetch a symbol s_i from S.
Step 3) Partition the interval I according to $p(\mathbf{A})$ and $p(\mathbf{B})$.

In Fig. 1(a), we show the interval partitioning on $I = [0, 1)$, where $p(\mathbf{A}) = 2/3$ and $p(\mathbf{B}) = 1/3$. In the case that I consists of two disjoint intervals I_1 and I_2, the partitioning is similar, as shown in Fig. 1(b), where $I_1 = [0, 1/3)$ and $I_2 = [2/3, 1)$.

Step 4) Perform one step of interval splitting according to k_i.

In Fig. 1(c), we show the intervals arrangement after performing interval splitting when $I = [0, 1)$, where the split location is determined by k_i. It should be pointed out that k_i is a normalized value over the range of potential split locations, and thus, may not be the absolute split location. Therefore, a key value of, e.g., $k_i = 0.25$ would identify a split location at the center of the first half of the range of valid key positions associated with the ith symbol, but would not generally lie at the absolute position 0.25. Due to the splitting operation, the portion of the $\mathbf{A}$ interval which is on the right of the split location will be moved to the right of the $\mathbf{B}$ interval. In the case that I consists of two disjoint intervals, the operation is similar, as shown in Fig. 1(d). It should be noted that the interval splitting is conducted with a constraint that each symbol sequence is represented by at most two distinct intervals. This requirement is to ensure that the coding efficiency loss of the ISAC is bounded by 1 bit per N-symbol sequence compared with a traditional AC.

Step 5) Update the interval I according to s_i, and increment $i = i + 1$.

For example, in Fig. 1(c), if $s_1 = \mathbf{A}$, then we update $I = [x_1, x_2) \bigcup [x_3, x_4)$.

Step 6) Repeat Steps 2)–5) until all the symbols are encoded.
Step 7) If I is continuous, then generate the final bit stream using interval I in a traditional manner. Otherwise if $I = I_1 \bigcup I_2$, then select the longer interval of I_1 and I_2, and output the final bit stream accordingly.

Assuming prefix-free coding, the codeword length N_c of the ISAC will lie in the range

$$\lceil -N \log_2 p(\mathbf{A}) \rceil \leq N_c \leq \lceil -N \log_2 p(\mathbf{B}) \rceil + 2. \qquad (1)$$

For more details please refer to [4].

B. SAC

In order to increase the security of the ISAC, the SAC was proposed, where the major difference from the ISAC is that two permutation steps are applied to the input symbol sequence and the output codeword, respectively [1]. Let $S = s_1 s_2 \cdots s_N$ be the symbol sequence to be encoded. The encoding procedure of the SAC is shown as follows.

Step 1) Map the sequence S into a block having four columns and $\lceil N/4 \rceil$ rows.
Step 2) Perform two key-driven cyclical shift steps to the resulting symbol block, and read out the data in raster-order to obtain the permuted symbol sequence $\tilde{S}$.

In Fig. 2, we show an example of the cyclical shift steps, where S is of length 16 and the key vectors controlling the

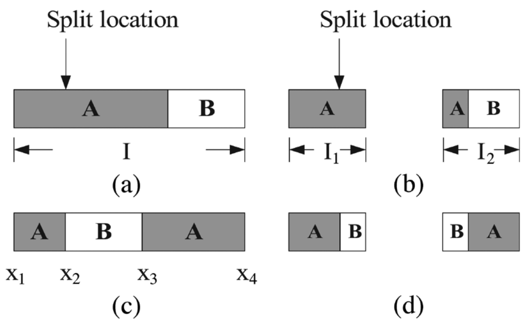

Fig. 1. Illustration of interval partitioning and interval splitting.

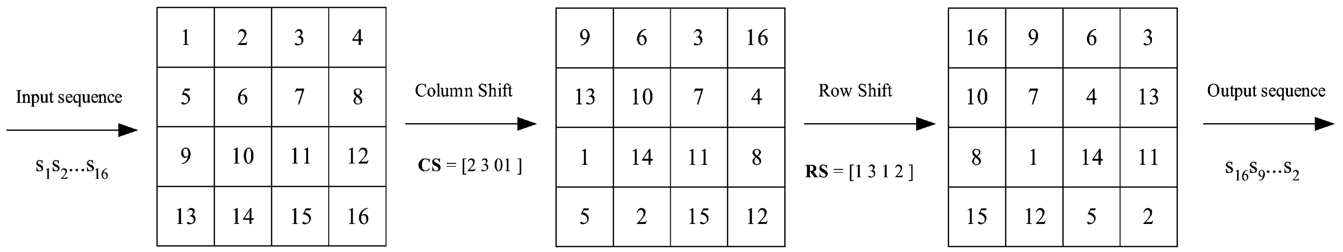

Fig. 2. Cyclical shifts applied to the input symbol sequence $S = s_1s_2\cdots s_{16}$.

column and the row shift offsets are $\mathbf{CS} = [2\ 3\ 0\ 1]$ and $\mathbf{RS} = [1\ 3\ 1\ 2]$, respectively.

Step 3) Input $\tilde{S}$ to the ISAC encoder and obtain the intermediate codeword $C = c_1c_2\cdots c_{N_c}$.

Step 4) Set $\hat{C} = c_1c_2\cdots c_{N_c-4}$ by removing the last four bits of C. Map $\hat{C}$ into a block having four columns and $\lceil (N_c-4)/4 \rceil$ rows.

Step 5) Perform the first round of shifts to the resulting bit block, which consists of two key-driven cyclical shift steps, one operating on columns and the other on rows. Here, the key vectors controlling the shift offsets depend on the last four bits of C, namely, $c_{N_c-3}c_{N_c-2}c_{N_c-1}c_{N_c}$.

Step 6) Reappend $c_{N_c-3}c_{N_c-2}c_{N_c-1}c_{N_c}$ to the resulting bit block.

Step 7) Perform the second round of shifts to the resulting bit block, which consists of two key-driven cyclical shift steps, one operating on columns and the other on rows. Here, the key vectors controlling the shift offsets are fixed for all C.

Step 8) Read out the data in raster-order from the resulting block to obtain the final bit stream.

It should be noted that the key streams used in the above steps are generated from a key scheduler using repeated XOR operation. For more details, please refer to [1].

III. Adaptive Chosen-Ciphertext Attack on the SAC

Following Kerckhoff's principle [20], the strength of a cryptosystem depends only on the key and, in particular, the security does not depend on keeping the encryption algorithm secret. This principle implies that the attacker knows the protocols and the overall system in which the cryptosystem is used, while only does not know the secret key. According to the information that the attacker can have access to, the attacks can be classified into several types [20]. A *ciphertext-only attack* is one where the attacker tries to deduce the secret key or the plaintext by only observing the ciphertext [20]. A *known-plaintext attack* is one where the attacker has a quantity of plaintext and the corresponding ciphertext [20]. A *chosen-plaintext attack* is one where the attacker chooses plaintext and is then given the corresponding ciphertext [20]. A *chosen-ciphertext attack* is one where the attacker can have access to the decoder, and thus can select ciphertext and obtain the corresponding plaintext [20]. An *adaptive chosen-ciphertext attack* is a chosen-ciphertext attack where the choice of ciphertext may depend on the plaintext received from previous requests [20].

In this section, we evaluate the security of the SAC using an adaptive chosen-ciphertext attack. More specifically, we adaptively choose codewords fed into the SAC decoder, and obtain the corresponding symbol sequences. Based on the relationship between the codewords and the symbol sequences, our objective is to recover the key vectors used in the codeword permutation step, which consists of two rounds of shifts. This is sufficient to break the SAC since after getting these key vectors, we can remove the codeword permutation step, and the resulting system has already been shown to be insecure in the original paper [1]. It should be pointed out that in practical applications, e.g., JPEG 2000, when a bit stream is decoded, only the reconstructed image data is available, while the output of the entropy decoder is not directly accessible. In this case, we need to perform some reverse operations to get the equivalent output of the entropy decoder. Take the JPEG 2000 for example. Let the original image data be IM, the compressed image data be C, and the reconstructed image data be R, respectively, as shown in Fig. 3. In the JPEG 2000, there are two types of compression: lossless and lossy compression [8]. In the lossless mode, the process of both the encoder and the decoder has no information loss, namely, in both the transform and the quantization stages, the information is perfectly preserved. Therefore, we can apply the forward transform and the quantization to the reconstructed image data R, so as to get the output of the entropy decoder. In the case of lossy compression, we can still apply the forward transform and the quantization to the reconstructed image data R, and get an approximation of the output of the entropy decoder. Provided that R is of high quality, this approximation is still good. It should be noted that in [1], it was also assumed that the attacker can have access to the input and the output of the SAC encoder, when the chosen-plaintext attack was applied.

For fixed splitting key vector $\mathbf{K}$, fixed key vectors used in the codeword permutation step, and fixed symbol sequence length N, the number of possible codewords generated by the SAC is 2^N, corresponding to 2^N different symbol sequences. Denote $\mathcal{C}$ as the set consisting of all these 2^N codewords. On the other hand, the lengths of the codewords generated by the SAC are within the range shown in (1). It can be seen that the number of all possible codewords whose lengths satisfying (1) is

$$\sum_{i=\lceil -N\log_2 p(\mathbf{A})\rceil}^{\lceil -N\log_2 p(\mathbf{B})\rceil+2} 2^i \tag{2}$$

which is certainly larger than 2^N. A natural question arising is, *What will the decoder do if we input a codeword which has length satisfying* (1) *while does not belong to the set* $\mathcal{C}$? Regarding this point, we make the following assumption.

Assumption 1: Prior to the decoding of any codeword C, we assume that the decoder has been informed the number of sym-

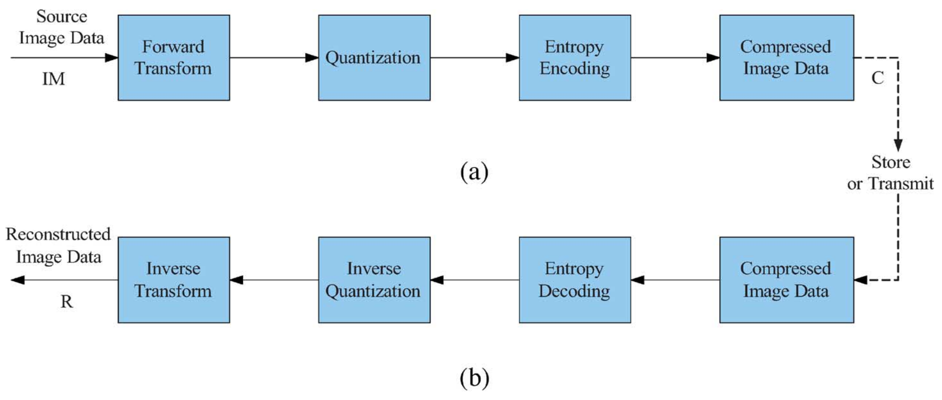

Fig. 3. General block diagram of the JPEG 2000: (a) encoder and (b) decoder.

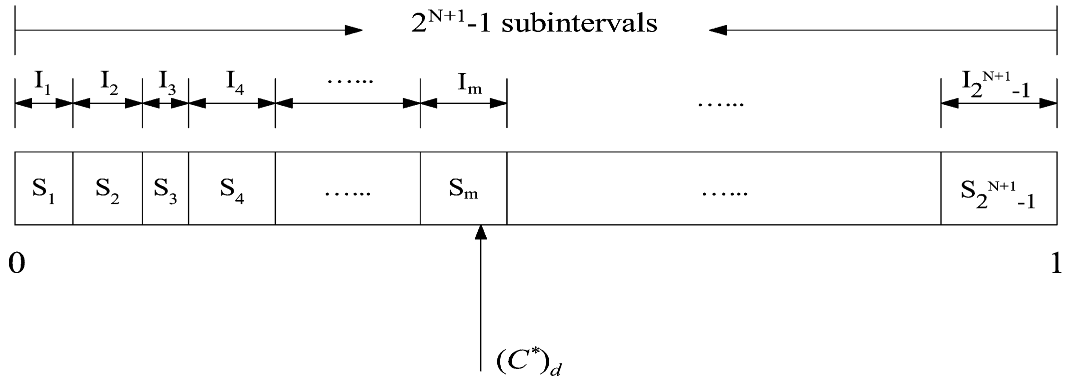
$2^{N+1}-1$ subintervals
I_1 I_2 I_3 I_4 …… I_m …… $I_{2^{N+1}-1}$
S_1 S_2 S_3 S_4 …… S_m …… $S_{2^{N+1}-1}$
0
1
$(C^*)_d$

Fig. 4. Decode a corrupted codeword based on the interval its decimal representation lies.

bols N it should decode. In addition, for any codeword C which has length satisfying (1) while does not belong to the set $\mathcal{C}$, the SAC decoder will treat it as a corrupted version of a codeword in $\mathcal{C}$, and will decode it into a length-N symbol sequence according to the interval that $(C)_d$ lies.

Remark: The decoding of the corrupted codewords can also be treated as an error-correction operation of the decoder. Due to the interval splitting, the interval [0,1) is partitioned into $2^{N+1}-1$ subintervals, each of which represents a symbol sequence of length N, as shown in Fig. 4. Let C^* be a codeword to be decoded, and I_m be the subinterval that the decimal representation $(C^*)_d$ lies. Let also S_m be the corresponding length-N symbol sequence associated with I_m. Then the decoder returns S_m as the decoded symbol sequence of C^*, irrespective whether $C^* \in \mathcal{C}$ or not. It should be pointed out that there are some other error-correcting AC schemes, e.g., AC with forbidden symbol [22], [23] and AC with soft resynchronization markers [24]. In fact, the SAC may conveniently be incorporated with a forbidden symbol to enable error detection and error correction capability.

In order for better illustration, in the following Section III-A, we first consider a simplified case using static key vectors, where both the first round and the second round of codeword cyclical shifts do not depend on the input codeword. In other words, the same key vectors are applied for all results. We then in Section III-B consider the case using adaptive key vectors, where one round of codeword cyclical shifts are input-dependent. We show that our method in Section III-A still works subject to some appropriate modifications. In Section III-C, we briefly discuss the feasibility of our attack under the situation that the Assumption 1 becomes invalid.

A. Static Key Vectors Used in the Codeword Permutation Step

In order to recover the key vectors used in the codeword permutation step, it suffices to find the correspondence of bit locations before and after the codeword permutation step.

Before going into the details, let us briefly outline the core idea of our method. Suppose we have already known that the second, the fourth, and the sixth bits in a codeword of length N_c become the last three bits after the codeword permutation step. We now wish to find which bit becomes the (N_c-3)th bit after the permutation step. We set $C = c_1 c_2 \cdots c_{N_c}$, where

$$c_i = \begin{cases} 1 & \text{for } i \in \mathcal{I} = \{2,4,6\} \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Define a set $\mathcal{A} = \{a | a \in \mathcal{Z}^+ \text{ and } 1 \le a \le N_c\}$. We then set $C'(j) = c'_1 c'_2 \cdots c'_{N_c}$, for $j \in \mathcal{A} \setminus \mathcal{I}$, where

$$c'_i = \begin{cases} 1, & \text{for } i = j \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

For example, if $N_c = 8$, then $C = 01010100$ and $C'(1) = 10000000$. After the codeword permutation step, the permuted versions of C and $C'(j)$ become $\tilde{C} = 00\cdots0111$ and

$$\tilde{C}'(j) = \underbrace{0\cdots01}_{\tilde{j}}0\cdots0$$

where $\tilde{j}$ is the location of the jth bit after the permutation step. If it happens that $\tilde{j} = N_c - 3$, then

$(\tilde{C}'(j))_d - (\tilde{C})_d = (00\cdots01)_d = 2^{-N_c}$, which is the minimum distance between any two distinct binary numbers of length N_c. On the contrary, if $\tilde{j} \neq N_c - 3$, then $(\tilde{C}'(j))_d - (\tilde{C})_d \geq (0\cdots01001)_d = 2^{-N_c} + 2^{-N_c+|\mathcal{I}|}$, where $\mathcal{I} = \{2,4,6\}$ and $|\cdot|$ denotes the cardinality of a set. This leads to the fact that if $\tilde{j} = N_c - 3$, then $S(C)$ and $S(C'(j))$ are very similar, even identical provided that N_c is sufficiently large. While if $\tilde{j} \neq N_c - 3$, then it is very likely that $S(C)$ and $S(C'(j))$ differ in many symbols. It is worthwhile to point out that in the case of $\tilde{j} \neq N_c - 3$, the minimum distance between $(\tilde{C}'(j))_d$ and $(\tilde{C})_d$ increases exponentially with the increasing $|\mathcal{I}|$. In this way, we can determine which bit becomes the $(N_c - 3)$th bit after the permutation step.

Algorithm 1: Find the correspondence of the bit locations before and after the codeword permutation step

```
Set $C^0 = 00\cdots0$ and $C^1 = 10\cdots00$, both of which have length $N_{c,\max} = \lceil -N\log_2 p(\mathbf{B})\rceil + 2$. Set $\mathcal{A} = \{a | a \in \mathcal{Z}^+$ and $1 \leq a \leq N_{c,\max}\}$. Initialize $m = 1$ and $n = 0$.
for $j = 1$ to $N_{c,\max}$ do
    Decode $C^j$ into $S(C^j)$.
    $C^{j+1} = C^j \gg 1$.
end for
repeat
    $\mathcal{I}_m = \varnothing$.
    Decode $C^0$ into $S(C^0)$.
    for $j \in \mathcal{A} \setminus \bigcup_{k=1}^{m} \mathcal{I}_k$ do
        if $S(C^j) = S(C^0)$ then
            $\mathcal{I}_m = \mathcal{I}_m \bigcup \{j\}$.
        end if
    end for
    Update $C^0 = c_1c_2\cdots c_{N_{c,\max}}$ where
        $$c_i = \begin{cases} 1, & \text{for } i \in \bigcup_{k=1}^{m}\mathcal{I}_k \\ 0, & \text{otherwise.} \end{cases} \quad (5)$$
    $m = m + 1, n = n + |\mathcal{I}_m|$.
until $n = N_{c,\max}$
Output: $\{\mathcal{I}_m\}, \{\mathcal{I}_{m-1}\}, \cdots, \{\mathcal{I}_1\}$, where $\{\cdot\}$ denotes the element of the corresponding set.
```

Aiming at finding the correspondence of the bit locations before and after the permutation step, we can use Algorithm 1 shown below. A brief explanation of Algorithm 1 is as follows. For $C^0 = 00\cdots0$ having length $N_{c,\max} = \lceil -N\log_2 p(\mathbf{B})\rceil + 2$, it is clear that its permuted version is $\tilde{C}^0 = 00\cdots0$, which corresponds to the leftmost interval. For

$$C^j = \underbrace{0\cdots01}_{j}0\cdots0$$

having length $N_{c,\max}$, the bit "1" will move to a random location after the codeword permutation step. Hence, $(\tilde{C}^j)_d \in \{2^{-N_{c,\max}}, 2^{-N_{c,\max}+1}, \cdots, 2^{-1}\}$, where $\tilde{C}^j$ is the permuted version of C^j. If for a certain j, we observe $S(C^j) = S(C^0)$, then we can determine that $\tilde{C}^j$ and $\tilde{C}^0$ are both within the leftmost interval. Therefore, if we group all these js such that $S(C^j) = S(C^0)$ into a set $\mathcal{I}_1$, we can see that the elements in $\mathcal{I}_1$ correspond to the indexes of the bits that become the last $|\mathcal{I}_1|$ bits after the codeword permutation step. Then we update C^0 by fixing the bits whose indexes belong to the set $\mathcal{I}_1$ to "1" and setting all the other bits to zero. By applying a very similar approach and repeating these steps, we can obtain the correspondence of the remaining bit locations before and after the codeword permutation step. Therefore, the index sequence $\{\mathcal{I}_M\}, \{\mathcal{I}_{M-1}\}, \cdots, \{\mathcal{I}_1\}$ would be the bit indexes after the codeword permutation step, where $\{\cdot\}$ denotes the elements of the corresponding set and M is the number of sets obtained using Algorithm 1.

It should be noted that for small m, especially $m = 1$, the cardinality of $\mathcal{I}_m$ may be larger than one, since multiple js satisfy $S(C^j) = S(C^0)$. This results in the fact that the index sequence we obtain by using Algorithm 1 may not be unique, which may lead to multiple key vectors. Aiming at determining the ordering of the elements in these sets, or equivalently deriving the key vectors used in the codeword permutation step, a simple way is to utilize the property of the cyclical shift, and the fact that the last four bits of the result after the first round of shift are identical to those of the final result after the whole codeword permutation step (see Fig. 5 below as an example). More specifically, after getting $\mathcal{I}_i$, for $1 \leq i \leq M$, we choose $N_c = N_{c,\max} - |\mathcal{I}_1| + 1$ and perform Algorithm 1 by simply replacing $N_{c,\max}$ with N_c. Denote the obtained sets as $\mathcal{I}'_i$, for $1 \leq i \leq M$. Since $N_c = N_{c,\max} - |\mathcal{I}_1| + 1$, we can easily find that $|\mathcal{I}'_1| = 1$, which means that $\mathcal{I}'_1$ is uniquely determined without ambiguity. Notice that the $\{\mathcal{I}'_1\}$th bit also corresponds to the last bit after the first round of shift, due to the bit removing and reappending operations. Thus, we can derive a column shift offset and a row shift offset for the first round of shift based on the correspondence of the $\{\mathcal{I}'_1\}$th bit. Following the same idea, we can increase N_c gradually to $N_c = N_{c,\max} - |\mathcal{I}_1| + 2$, $N_c = N_{c,\max} - |\mathcal{I}_1| + 3$, and $N_c = N_{c,\max} - |\mathcal{I}_1| + 4$, and perform Algorithm 1. We can get four column shift offsets and row shift offsets for the first round of shift. Based on these four pairs of shift offsets, together with the correspondence provided by the other $\mathcal{I}_i$, it is straightforward to derive the whole key vectors.

In the following, we give a concrete example of applying Algorithm 1 with the following configurations.

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

$$\mathbf{K} = \left[\frac{5}{16}\ \frac{5}{16}\ \frac{1}{16}\ \frac{7}{16}\ \frac{11}{16}\ \frac{15}{16}\ \frac{9}{16}\ \frac{7}{16}\ \frac{5}{16}\ \frac{13}{16}\ \frac{3}{16}\right.$$
$$\left.\frac{15}{16}\ \frac{9}{16}\ \frac{11}{16}\ \frac{1}{16}\ \frac{13}{16}\right] \quad (6)$$

- The key vectors used in the symbol permutation step are

$$\mathbf{RS} = [RS_1\ RS_2\ RS_3\ RS_4] = [1\ 2\ 1\ 3]$$
$$\mathbf{CS} = [CS_1\ CS_2\ CS_3\ CS_4] = [0\ 3\ 1\ 2] \quad (7)$$

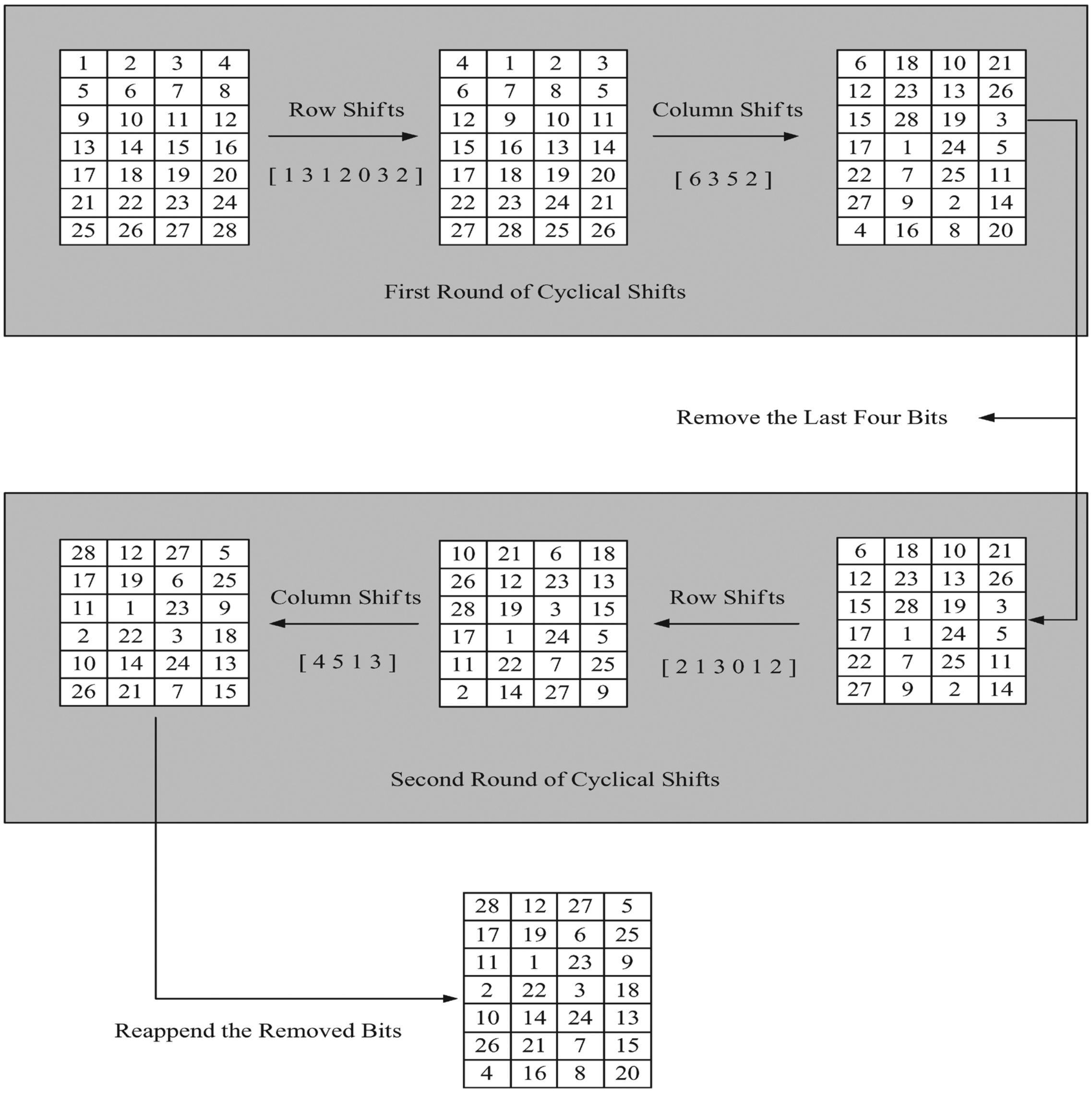
Row Shifts
[1 3 1 2 0 3 2]
Column Shifts
[6 3 5 2]
First Round of Cyclical Shifts
Remove the Last Four Bits
Row Shifts
[2 1 3 0 1 2]
Column Shifts
[4 5 1 3]
Second Round of Cyclical Shifts
Reappend the Removed Bits

Fig. 5. Codeword permutation step using the key vectors shown in (8), 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 permutation step.

- The key vectors used in the codeword permutation step are

$$\begin{aligned}\mathbf{RC}_1 &= [RC_{1,1}\ RC_{1,2}\ RC_{1,3}\ RC_{1,4}\ RC_{1,5}\ RC_{1,6}\ RC_{1,7}]\\ &= [1\ 3\ 1\ 2\ 0\ 3\ 2]\\ \mathbf{CC}_1 &= [CC_{1,1}\ CC_{1,2}\ CC_{1,3}\ CC_{1,4}] = [6\ 3\ 5\ 2]\\ \mathbf{RC}_2 &= [RC_{2,1}\ RC_{2,2}\ RC_{2,3}\ RC_{2,4}\ RC_{2,5}\ RC_{2,6}]\\ &= [2\ 1\ 3\ 0\ 1\ 2]\\ \mathbf{CC}_2 &= [CC_{2,1}\ CC_{2,2}\ CC_{2,3}\ CC_{2,4}] = [4\ 5\ 1\ 3] \qquad (8)\end{aligned}$$

where $\mathbf{RC}_1$, $\mathbf{CC}_1$, $\mathbf{RC}_2$, and $\mathbf{CC}_2$ are used for the first round of row and column cyclical shifts, and the second round of row and column cyclical shifts, respectively. From the perspective of the decoder, in the first round, the codeword is subject to two cyclical shifts, one operating on the rows and the other on the columns, according to $\mathbf{RC}_1$ and $\mathbf{CC}_1$, respectively. After that, the last four bits of the result are removed, and the remaining bits are subject to another round of cyclical shifts using $\mathbf{RC}_2$, and $\mathbf{CC}_2$, respectively. Finally, the previously removed four bits are reappended and the data are read out in raster-order to form the codeword fed into the ISAC decoder. The permutation procedure can also be illustrated in Fig. 5. It should be noted that, in this subsection, the key vectors used in the codeword permutation step are assumed to be fixed, and hence, they are input-independent.

After applying Algorithm 1, we get $\mathcal{I}_1 = \{4, 7, 8, 13, 14, 15, 16, 20, 21, 24, 26\}$, $\mathcal{I}_2 = \{10, 18\}$, $\mathcal{I}_3 = \{3\}$, $\mathcal{I}_4 = \{22\}$, $\mathcal{I}_5 = \{2\}$, $\mathcal{I}_6 = \{9\}$, $\mathcal{I}_7 = \{23\}$, $\mathcal{I}_8 = \{1\}$, $\mathcal{I}_9 = \{11\}$, $\mathcal{I}_{10} = \{25\}$, $\mathcal{I}_{11} = \{6\}$, $\mathcal{I}_{12} = \{19\}$, $\mathcal{I}_{13} = \{17\}$, $\mathcal{I}_{14} = \{5\}$, $\mathcal{I}_{15} = \{27\}$, $\mathcal{I}_{16} = \{12\}$, and $\mathcal{I}_{17} = \{28\}$. Equivalently, we can get the result shown in Fig. 6, where $X_i \in \mathcal{I}_2$ and $Y_i \in \mathcal{I}_1$.

Since the cardinality of $\mathcal{I}_1$ and $\mathcal{I}_2$ is larger than one, we still need to determine the ordering of their elements, or equivalently derive the key vectors used in the codeword permutation step. As stated previously, we can choose $N_c = N_{c,\max} - |\mathcal{I}_1| + 1$

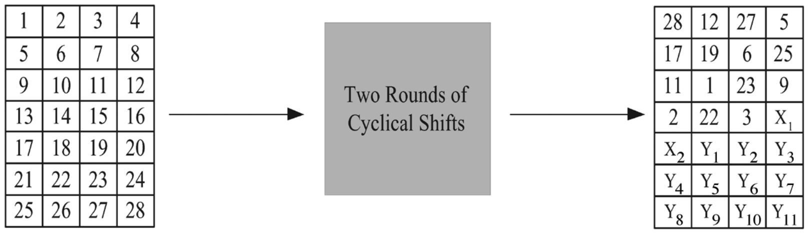

Fig. 6. Correspondence of bit locations before and after the codeword permutation step.

and perform Algorithm 1 by simply replacing $N_{c,\max}$ with N_c. Using the configurations shown in this example, we can find that $\mathcal{I}_1' = \{7\}$, which implies that the 7th bit becomes the $N_c = N_{c,\max} - |\mathcal{I}_1| + 1 = 18$th bit after the first round of shifts. Notice that the 7th bit originally locates at the second row and the third column. We can then determine that $RC_{1,2} = 3$ and $CC_{1,2} = 3$. By increasing N_c gradually to $N_c = N_{c,\max} - |\mathcal{I}_1| + 2$, $N_c = N_{c,\max} - |\mathcal{I}_1| + 3$, and $N_c = N_{c,\max} - |\mathcal{I}_1| + 4$, and performing Algorithm 1, we can get $RC_{1,5} = 0$ and $CC_{1,3} = 5$; $RC_{1,3} = 1$ and $CC_{1,4} = 2$; and $CC_{1,1} = 6$. Based on these shift offsets, together with the bit correspondence shown in Fig. 6, deriving the whole key vectors becomes a trivial issue.

Now we roughly calculate the complexity involved in our proposed method, where the complexity is measured by the number of decoding operations. Using Algorithm 1, we need approximately $2N_{c,\max}$ decoding operations. Since Algorithm 1 is used for five times, corresponding to $N_c = N_{c,\max}$ and $N_c = N_{c,\max} - |\mathcal{I}_1| + i$, for $1 \le i \le 4$, the overall complexity is about $10N_{c,\max}$ decoding operations, which is of order $\mathcal{O}(N)$.

B. Adaptive Key Vectors Used in the Codeword Permutation Step

In this subsection, we consider the case using adaptive key vectors in the codeword permutation step. Recall in Section III-A, the key vectors used in the two rounds of shifts are fixed for all input codewords. However, in the current case, the key vectors used in the second round of shifts depend on the last four bits after performing the first round of shifts. For example, suppose Fig. 5 shows the codeword permutation procedure for the case of adaptive key vectors, then $\mathbf{RC}_2$ and $\mathbf{CC}_2$ would depend on the fourth, the sixteenth, the eighth, and the twentieth bits of the input codeword C.

To deal with this case, we first face the problem of determining the locations of the bits that become the last four bits after the first round of shifts. Let the input codeword length be N_c. The complexity of exhaustively searching these four bits is $\binom{N_c}{4}4!$, which is of order $\mathcal{O}(N_c^4)$. A more efficient way to achieve this goal is to decode $C^0 = 00\cdots0$ and

$$C^j = \underbrace{0\cdots01}_{j}0\cdots0$$

for $1 \le j \le N_c$, and obtain $\mathcal{I}_1$ in a similar way as done in Algorithm 1. It can be easily seen that these four bits belong to $\mathcal{I}_1$, provided that $|\mathcal{I}_1| \ge 4$. This reduces the complexity of exhaustive searching to $\binom{|\mathcal{I}_1|}{4}4!$. An even better way to this end is to utilize the fact that the last four bits after the first round of shifts are the same as those of the final result after the whole codeword permutation step. Hence, by first reducing the input codeword length, and then gradually increasing it, as stated in Section III-A, we can find these four bits with complexity $\mathcal{O}(N_c)$. For the sake of conciseness, we omit the details here.

Since the second round of shifts depend on the last four bits of the result after the first round of shifts, there are $2^4 = 16$ groups of key vectors that should be recovered. Suppose now we wish to recover the key vectors for the case that the last four bits after the first round of shifts are d_1, d_2, d_3, and d_4, respectively. As we have already known the locations of the bits that eventually become the last four bits after the first round of shifts, we can fix the bits in these locations to be d_1, d_2, d_3, and d_4, respectively. Therefore, all the key vectors used in the codeword permutation step become fixed.

Notice that, after fixing four bits, the minimum decimal distance between any two distinct binary numbers of length $N_{c,\max}$ becomes $2^{-N_{c,\max}+4}$, instead of $2^{-N_{c,\max}}$. Recall in Algorithm 1, after determining $\mathcal{I}_1$, our method is based on the fact that if the jth bit happens to be what we wish to find, then the constructed C^0 and C^j satisfy $(\tilde{C^j})_d = (\tilde{C^0})_d + 2^{-N_{c,\max}}$, where $\tilde{C^0}$ and $\tilde{C^j}$ are the permuted versions of C^0 and C^j, respectively. This results in the fact that $S(C^0)$ and $S(C^j)$ are very likely identical. Therefore, in the current case where we fix four bits, if $(\tilde{C^0})_d + 2^{-N_{c,\max}+4}$ is still within the same interval as that of $(\tilde{C^0})_d$, then our method in Section III-A can be directly applied.

Let $I_{\tilde{C}}$ and $I_{\tilde{C}+\Delta}$ be the intervals associated with $(\tilde{C})_d$ and $(\tilde{C})_d + 2^{-N_{c,\max}+4}$, respectively. The probability $\Pr[I_{\tilde{C}} = I_{\tilde{C}+\Delta}]$ can be roughly approximated by $\Pr[\text{The width of } I_{\tilde{C}} > 2^{-N_{c,\max}+4}]$. Without splitting, the width of $I_{\tilde{C}}$ is $p(\mathbf{A})^{N(\mathbf{A})} \cdot p(\mathbf{B})^{N-N(\mathbf{A})}$, where N is the symbol sequence length and $N(\mathbf{A})$ is the number of $\mathbf{A}$s in the symbol sequence corresponding to $\tilde{C}$. Due to the interval splitting, each interval except one is split into two, using a uniformly distributed splitting key vector [1]. We can estimate the width of each interval after splitting as $1/2 \cdot p(\mathbf{A})^{N(\mathbf{A})} \cdot p(\mathbf{B})^{N-N(\mathbf{A})}$, noticing the uniform distribution of the splitting key vector. Therefore

$$\begin{aligned}
&\Pr[I_{\tilde{C}} = I_{\tilde{C}+\Delta}] \\
&\simeq \Pr[\text{The width of } I_{\tilde{C}} > 2^{-N_{c,\max}+4}] \\
&\simeq \Pr\left[1/2 \cdot p(\mathbf{A})^{N(\mathbf{A})} \cdot p(\mathbf{B})^{N-N(\mathbf{A})} > 2^{-N_{c,\max}+4}\right] \\
&= \Pr\left[N(\mathbf{A}) > \frac{5 - N_{c,\max} - N\log_2 p(\mathbf{B})}{\log_2 p(\mathbf{A})/p(\mathbf{B})}\right] \\
&= \Pr\left[N(\mathbf{A}) > \frac{3 - \lceil -N\log_2 p(\mathbf{B})\rceil - N\log_2 p(\mathbf{B})}{\log_2 p(\mathbf{A})/p(\mathbf{B})}\right] \quad (9)
\end{aligned}$$

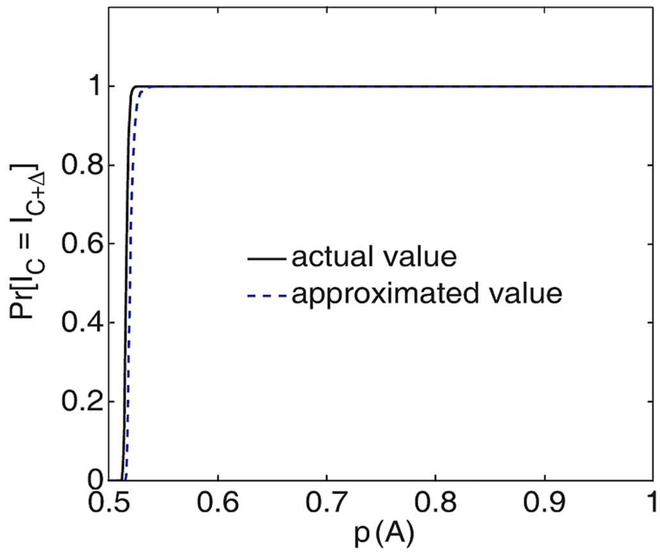

Fig. 7. Comparison between the actual value and the approximated value of $\Pr[I_{\tilde{C}} = I_{\tilde{C}+\Delta}]$, where $N = 64$.

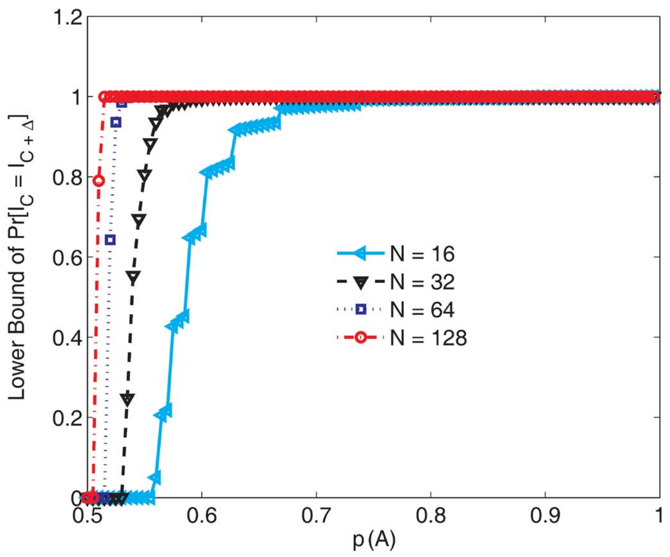

Fig. 8. Lower bound of $\Pr[I_{\tilde{C}} = I_{\tilde{C}+\Delta}]$ shown in (11) versus $p(\mathbf{A})$, for different N.

$$\begin{aligned} &\geq \Pr\left[N(\mathbf{A}) > \frac{3}{\log_2 p(\mathbf{A})/p(\mathbf{B})}\right] \\ &= \Pr\left[N(\mathbf{A}) > \left\lfloor \frac{3}{\log_2 p(\mathbf{A})/p(\mathbf{B})} \right\rfloor\right]. \end{aligned} \tag{10}$$

Letting $\tau = 3/(\log_2 p(\mathbf{A})/p(\mathbf{B}))$, we have the following proposition concerning $\Pr[N(\mathbf{A}) > \lfloor \tau \rfloor]$.

Proposition 1: The following lower bound holds

$$\Pr\left[N(\mathbf{A}) > \lfloor \tau \rfloor\right] \geq \begin{cases} 1 - \exp\left\{-\frac{1}{2p(\mathbf{A})} \frac{(p(\mathbf{A})N - \lfloor \tau \rfloor)^2}{N}\right\}, & \text{if } \lfloor \tau \rfloor \leq Np(\mathbf{A}) \\ 0, & \text{otherwise.} \end{cases} \tag{11}$$

Proof: See the Appendix. □

Remarks:

- We can use the lower bound shown in (11) as a conservative approximation of $\Pr[I_{\tilde{C}} = I_{\tilde{C}+\Delta}]$. In Fig. 7, we show the comparison of this approximated probability and the actual probability calculated from (9), in the case that $N = 64$. It can be seen that the lower bound shown in (11) is a good approximation of the actual probability.
- Let $f(p(\mathbf{A}), N) \triangleq 1 - \exp\{-(1/2p(\mathbf{A}))(((p(\mathbf{A})N - \lfloor \tau \rfloor)^2)/N)\}$, and $p^*(\mathbf{A})$ be the solution to the equation $\lfloor \tau \rfloor = Np(\mathbf{A})$. Clearly, as $p(\mathbf{A}) \rightarrow p^*(\mathbf{A})$, $f(p(\mathbf{A}), N) \rightarrow 0$. In addition, as $p(\mathbf{A}) \rightarrow 1$, $f(p(\mathbf{A}), N) \rightarrow 1$ for large N. Notice

$$-\frac{1}{2p(\mathbf{A})} \frac{(p(\mathbf{A})N - \lfloor \tau \rfloor)^2}{N} = -\frac{1}{2}p(\mathbf{A})N + \lfloor \tau \rfloor - \frac{\lfloor \tau \rfloor^2}{2p(\mathbf{A})N}. \tag{12}$$

It can be observed that the third term of (12) becomes vanishingly small as $p(\mathbf{A})$ increases from $p^*(\mathbf{A})$, especially for large N. Then, (12) is dominated by the first two terms. As $p(\mathbf{A})$ increases, then τ decreases, which leads to the fact that $f(p(\mathbf{A}), N)$ increases exponentially with respect to the increasing $p(\mathbf{A})$. This implies that the transition period from $f(p(\mathbf{A}), N) \rightarrow 0$ to $f(p(\mathbf{A}), N) \rightarrow 1$ is very short, which can be illustrated in Fig. 8. It should be also noted that the transition period is even shorter for larger N. In the SAC, the symbol sequence length N should be sufficiently large to ensure large enough key space, which precludes the brute-force attack. For example $N = 128$ could be used. In this case, from Fig. 8, we can see that $\Pr[I_{\tilde{C}} = I_{\tilde{C}+\Delta}]$ is close to 1 for a wide range of $p(\mathbf{A})$. This implies that the method described in Section III-A is still very likely applicable here.
- It should be pointed out that the range of $p(\mathbf{A})$ satisfying $\lfloor \tau \rfloor > Np(\mathbf{A})$ is very narrow for large N. When $\lfloor \tau \rfloor > Np(\mathbf{A})$, we have

$$\tau \geq \lfloor \tau \rfloor > Np(\mathbf{A}) \geq \frac{N}{2}. \tag{13}$$

Then, we find that the $p(\mathbf{A})$ satisfying $\lfloor \tau \rfloor > Np(\mathbf{A})$ belongs to a subset of the following range:

$$\frac{1}{2} \leq p(\mathbf{A}) < \frac{2^{6/N}}{1 + 2^{6/N}}. \tag{14}$$

In the case of $N = 128$, (14) becomes $0.5 \leq p(\mathbf{A}) < 0.5081$. This again implies that in a significantly wide range of $p(\mathbf{A})$, the probability $\Pr[I_{\tilde{C}} = I_{\tilde{C}+\Delta}]$ is close to 1.

In the following, we also give a simple example to calculate the probability $\Pr[I_{\tilde{C}} = I_{\tilde{C}+\Delta}]$ using (9) in the case that $p(\mathbf{A}) = 2/3$, $p(\mathbf{B}) = 1/3$, and $N = 16$. We have

$$\begin{aligned} \Pr[I_{\tilde{C}} = I_{\tilde{C}+\Delta}] &\simeq \Pr\left[N(\mathbf{A}) > 2.35\right] \\ &= 1 - \sum_{i=0}^{2} \Pr\left[N(\mathbf{A}) = i\right] \\ &= 1 - \sum_{i=0}^{2} \binom{N}{i} p(\mathbf{A})^i p(\mathbf{B})^{N-i} \\ &= 0.999988. \end{aligned} \tag{15}$$

In other words, the probability that $(\tilde{C})_d + 2^{-N_{c,\max}+4}$ and $(\tilde{C})_d$ belong to different intervals is approximately $1.2 \cdot 10^{-5}$. Hence, after fixing four bits, the method in Section III-A is still very likely applicable here. We also test the example shown in Section III-A by fixing the fourth, the sixteenth, the eighth, and the twentieth bits of the input codeword, and find that the result in Section III-A still holds.

C. What if Assumption 1 Becomes Invalid?

In this subsection, we consider the problem of recovering the key vectors used in the codeword permutation step when Assumption 1 stated above becomes invalid. More specifically, the decoder will check the validity of the input codewords. Let $\mathcal{C}$ be the set consisting of the 2^N codewords corresponding to the 2^N symbol sequences. If the input codeword $C \in \mathcal{C}$, then the decoder will output the corresponding symbol sequence. Otherwise, the decoder will only claim an error while not outputting any symbol sequence. The error-correcting AC with forbidden symbol can be interpreted as a special case of this, where the decoder skips the current coding pass when any error is detected [22], [23].

In this context, the problem of our method described in the previous subsections is that the input codewords may not be valid codewords, and thus, the corresponding symbol sequences could not be obtained. Since the set $\mathcal{C}$ depends on the splitting key vector $\mathbf{K}$ and the codeword permutation step, we do not know it in advance. A method to find valid codewords is to randomly search codewords with the most likely codeword length, noticing the fact that the codeword lengths in the range of (1) are not equiprobable. For a symbol sequence S consisting of n $\mathbf{A}$s and $N-n$ $\mathbf{B}$s, the codeword length is within the range

$$\begin{aligned}\lceil -(n\log_2 p(\mathbf{A}) + (N-n)\log_2 p(\mathbf{B}))\rceil &\le N_c \\ &\le \lceil -(n\log_2 p(\mathbf{A}) + (N-n)\log_2 p(\mathbf{B}))\rceil + 2. \quad (16)\end{aligned}$$

We can roughly estimate the codeword length of S as $l = \lceil -(n\log_2 p(\mathbf{A}) + (N-n)\log_2 p(\mathbf{B}))\rceil + 1$. Since the number of symbol sequences consisting of n $\mathbf{A}$s and $N-n$ $\mathbf{B}$s is $\binom{N}{n}$, we can estimate the percentage of valid codewords with length l among all the codewords having the same lengths as

$$\binom{N}{n}2^{-l} = \binom{N}{n}2^{-\lceil -(n\log_2 p(\mathbf{A}) + (N-n)\log_2 p(\mathbf{B}))\rceil - 1}. \quad (17)$$

Therefore, a good strategy is to search valid codewords with codeword length l corresponding to n that maximizes (17). In the case of $p(\mathbf{A}) = 2/3$ and $N = 16$, we find that $n = 11$ maximizes (17), which leads to $l = 16$ and the percentage of valid codewords is around 0.07. Notice the fact that two valid codewords which differ in the last several bits after the codeword permutation step have similar output symbol sequences. Based on this, we can find the correspondence of some bit locations before and after the codeword permutation step. Hence, we can reduce the key space of the key vectors used in the codeword permutation step, or even recover the whole key vectors given enough trials.

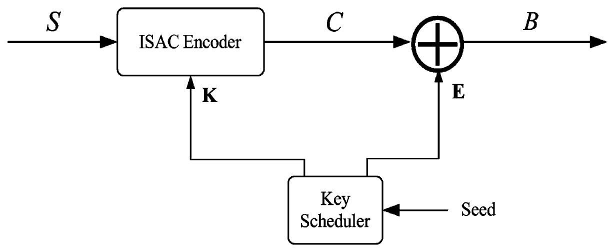

Fig. 9. Improved system of the SAC.

Another more efficient method to deal with this case is to change $p(\mathbf{A})$ and $p(\mathbf{B})$, provided that it is possible. Since the SAC is a non-adaptive AC, the encoder has to inform the decoder the source statistics, prior to the actual encoding/decoding. For example, the source statistics information can be embedded into the header. In this case, the attacker can change this header information in order to set any desired statistics. Alternatively, the encoder can embed some training data in the bit stream, and the decoder can obtain the source statistics from the training data. The latter strategy sometimes is also called the semi-adaptive coding [25]. In this case, the attacker can forge the training data so as to set any desired source statistics. According to (1), in order to make the codeword length range most compact, the attacker can set $p(\mathbf{A}) = p(\mathbf{B}) = 1/2$. Then there are only three possible codeword lengths: N, $N+1$ and $N+2$. Therefore, it is easy to find valid codewords, and apply a very similar method as stated in the previous subsections, to recover the key vectors used in the codeword permutation step.

IV. Joint Security and Performance Enhancement for the SAC

In Section III, we show that the SAC is vulnerable against an adaptive chosen-ciphertext attack. A natural question now arising is how to improve the SAC in order to resist this adaptive chosen-ciphertext attack. In addition, as also mentioned in the original paper [1], it is interesting to extend the SAC such that it can be conveniently incorporated with the context-based coding. In this section, we attempt to address these two problems simultaneously. In other words, our target is to suggest an improved version of the SAC such that 1) the system is immune against all the known attacks including the adaptive chosen-ciphertext attack described in Section III and 2) the system can utilize the full-context information so as to enable the context-based coding.

The schematic diagram of the improved system (encoder) is shown in Fig. 9, which mainly consists of two parts: 1) an ISAC encoder and 2) a key scheduler. Let the symbol sequence be $S = s_1 s_2 \cdots s_N$ that is to be encoded. The basic steps of performing the encoding are as follows.

Step 1) Encode S using an ISAC encoder with splitting key vector $\mathbf{K}$. Denote the generated bit stream as $C = c_1 c_2 \cdots c_{N_c}$, where N_c satisfies (1).

Step 2) Perform bit-wise XOR operation between C and a key stream $\mathbf{E}$, where $\mathbf{E}$ has the same length as C. In other words, the final bit stream $B = C \oplus \mathbf{E}$.

Compared with the original SAC, we remove the input symbol permutation step. In addition, we replace the output codeword permutation step with a simple bit-wise XOR step.

TABLE I
INPUT SYMBOL SEQUENCES AND THEIR CORRESPONDING CODEWORDS

Input sequence	Intermediate codeword C	Decimal of C	Final codeword B	Decimal of B
S_1: **B B B A B B B B B A B A A B B A**	1001000011000101101	0.5655	0011101001101111000	0.2283
S_2: **A A B A B B B B B A B A A A B B**	001100111101111	0.2026	100110010111010	0.5994
S_3: **B A B A B B B B B A B A A A B B**	0111111000010011101	0.4925	1101010010111001000	0.8309

The design of the key scheduler is very flexible; we can either use the keyed XOR operation as in [1] or other highly efficient pseudorandom number generators. The only private information of the improved system is the seed used in the key scheduler, which is assumed to be of length 128 bits, so as to ensure high enough level of security.

In the following subsections, we evaluate the security and the performance of the improved system. As mentioned in [1], the security analysis is a challenging job for any cryptosystem, since showing robust against known attacks does not preclude the other unknown attacks. Therefore, we evaluate the security under some known attacks, and show that it is secure against these attacks. More specifically, we use the ciphertext-only attack, the chosen-plaintext attack and the chosen-ciphertext attack.

A. Ciphertext-Only Attack

In this attack scenario, the attacker can only have access to the encrypted bit stream. Since the information available to the attacker is very limited, a very commonly used approach is the brute-force attack, whose complexity is related to the key space.

Since the only private information of the improved system is the seed used in the key scheduler and it is of length 128 bits, then the key space is 2^{128}, which can ensure satisfactory level of security for the digital rights management applications.

Alternatively, the attacker may wish to recover the key stream $\mathbf{K} = (k_1, k_2, \cdots k_N)$ used in the ISAC encoder and $\mathbf{E}$ used in the bit-wise XOR step. Suppose each k_i is of length U bits and the input symbol sequence length is N. The length of the key stream used in the ISAC encoder is then UN. Let the length of the generated bit stream be N_c, which satisfies (1). Thus, the total complexity of breaking the key stream is 2^{N_c+UN}. In order to make this complexity sufficiently large, we impose a constraint on the input sequence length, i.e.,

$$\lceil -N \log_2 p(\mathbf{A}) \rceil + UN > 128 \tag{18}$$

which ensures that $2^{N_c+UN} > 2^{128}$. In the case of $p(\mathbf{A}) = 2/3$, $p(\mathbf{B}) = 1/3$, and $U = 3$, we get $N \geq 36$. This requirement can be easily fulfilled in practice. Provided that (18) holds, the attacker would rather use the brute-force attack to break the seed used in the key scheduler. Therefore, we ensure that the key size is sufficiently large to preclude the brute-force attack.

Another large class of ciphertext-only attack is based on the analysis of statistical properties of the final bit stream B. It is thus important to investigate the statistics of B in order to evaluate the practical security of the improved system. Since the key stream used in the XOR step is generated from the key scheduler, it is reasonable to assume that the bits of $\mathbf{E}$ are nearly i.i.d. Since $B = C \oplus \mathbf{E}$, we know from [20] that B is also nearly i.i.d., since C and $\mathbf{E}$ are independently generated. Due to the fact that B is nearly i.i.d., the attacker can hardly find information about the secret key in the improved system only from the statistics of B. Thus, the improved system is secure against the ciphertext-only attack.

B. Chosen-Plaintext Attack

In this attack scenario, the attacker is allowed to input several symbol sequences to the encoder, and obtain the corresponding codewords. In [1], several chosen-plaintext attack methods were proposed to break the standalone ISAC and the hybrid system combining the ISAC with the codeword permutation step. It is thus important to investigate the security of the improved system under these attacks.

The first attack method described in [1] to break the standalone ISAC is to encode two-symbol input sequences $\mathbf{AA}$, $\mathbf{AB}$, $\mathbf{BA}$, and $\mathbf{BB}$, and construct a relationship between the split location u with the probabilities $p(\mathbf{A})$ and $p(\mathbf{B})$. For instance, if $C(\mathbf{BX}) < C(\mathbf{AX})$, then either $\mathbf{A}$ was split with $u < p(\mathbf{AA})/2$ or $\mathbf{B}$ was split with $u > p(\mathbf{A}) + p(\mathbf{BA}) + p(\mathbf{BB})/2$, where $C(S)$ is the encoded codeword of the symbol sequence S using the standalone ISAC encoder, and $\mathbf{X}$ can be either $\mathbf{A}$ or $\mathbf{B}$. Using this relationship, the attacker can significantly reduce the possible split locations. However, in the improved system, the attacker can only have access to $B = C \oplus \mathbf{E}$, instead of C. Therefore, even if $C(\mathbf{BX}) < C(\mathbf{AX})$, the inequality $B(\mathbf{BX}) < B(\mathbf{AX})$ may not be true, where $B(S) = C(S) \oplus \mathbf{E}$ for a certain symbol sequence S. For example, we suppose $C(\mathbf{BX}) = 00$ and $C(\mathbf{AX}) = 01$ for a certain $\mathbf{X}$. Assume that the key stream $\mathbf{E} = 11$. Then $B(\mathbf{BX}) > B(\mathbf{AX})$. While in the case of $\mathbf{E} = 00$, $B(\mathbf{BX}) < B(\mathbf{AX})$. It can be seen that these inequities depend on the key stream $\mathbf{E}$ as well. Hence, using this method, there is no clue to find the split locations, even in this two-symbol input sequences case.

Another method proposed in [1] to evaluate the security of the standalone ISAC is based on the following fact. Suppose the attacker finds three input symbol sequences S_1, S_2, and S_3 satisfying two conditions: 1) the first symbol of S_1 and S_3 is $\mathbf{A}$, and the first symbol of S_2 is $\mathbf{B}$; 2) $C(S_1) < C(S_2) < C(S_3)$. Then, it can be determined that symbol $\mathbf{A}$ is split by k_1, and the split location must be between $C(S_1)$ and $C(S_2)$. Nevertheless, this attack method cannot be successful for the improved system since the XOR operation changes the values of $C(S_1)$, $C(S_2)$, and $C(S_3)$, and hence, it is impossible to compare them without knowing the key stream $\mathbf{E}$. Table I gives an example of the input

TABLE II
RELATIONSHIP BETWEEN THE KEY STREAM $\mathbf{E}$ AND s_i^1. IN THE LAST ROW, $\mathbf{X}$ CAN BE EITHER $\mathbf{A}$ OR $\mathbf{B}$

	$\mathbf{E}=000$	$\mathbf{E}=001$	$\mathbf{E}=010$	$\mathbf{E}=011$	$\mathbf{E}=100$	$\mathbf{E}=101$	$\mathbf{E}=110$	$\mathbf{E}=111$
s_1^1	**A**	**A**	**A**	**B**	**B**	**A**	**A**	**A**
s_2^1	**A**	**A**	**B**	**A**	**A**	**B**	**A**	**A**
s_3^1	**A**	**B**	**A**	**A**	**A**	**A**	**B**	**A**
s_4^1	**B**	**A**	**A**	**A**	**A**	**A**	**A**	**B**
s_5^1	**B**	**A**	**A**	**A**	**A**	**A**	**A**	**B**
s_6^1	**A**	**B**	**A**	**A**	**A**	**A**	**B**	**A**
s_7^1	**A**	**A**	**B**	**A**	**A**	**B**	**A**	**A**
s_8^1	**A**	**A**	**A**	**B**	**B**	**A**	**A**	**A**
Symbol Split	**A**	**X**	**X**	**B**	**B**	**X**	**X**	**A**

symbol sequence and the output codeword pairs, where the input symbol sequence length $N = 16$, $p(\mathbf{A}) = 2/3$, $p(\mathbf{B}) = 1/3$, and the splitting key vector is

$$\mathbf{K} = \left[\frac{5}{16}\ \frac{5}{16}\ \frac{1}{16}\ \frac{7}{16}\ \frac{11}{16}\ \frac{15}{16}\ \frac{9}{16}\ \frac{7}{16}\ \frac{5}{16}\ \frac{13}{16}\ \frac{3}{16}\right.$$
$$\left.\frac{15}{16}\ \frac{9}{16}\ \frac{11}{16}\ \frac{1}{16}\ \frac{13}{16}\right]. \quad (19)$$

Since $k_1 = 5/16 < 1/2$, symbol $\mathbf{A}$ is split in the first step of encoding. However, after performing XOR operation to the codewords, $B(S_1) < B(S_2) < B(S_3)$ holds, which leads to the conclusion that $\mathbf{B}$ is split using the attack method in [1]. Therefore, this attack method cannot be successful against the improved system.

In [1], another advanced attack method was proposed to break the hybrid system combining the ISAC with the codeword permutation step. The basic idea is to search for the split symbol sequence according to the zeros and ones in the codeword relative to the whole codeword length. The success of this attack is due to the fact that the leftmost interval associated with the all-zeros codeword or the rightmost interval associated with the all-ones codeword correspond to the split symbol sequence [1]. In the improved system, since the final bit stream $B = C \oplus \mathbf{E}$, the number of zeros and ones in B depends on both C and $\mathbf{E}$, which individually is unknown for the attacker. On the other hand, the all-zeros codeword and the all-ones codeword are moved to random intervals associated with $\mathbf{E}$ and $\bar{\mathbf{E}}$, respectively. Therefore, from the percentage of the zeros and ones, it is difficult for the attacker to search for the split symbol sequence, which contains the information of the split location.

C. Chosen-Ciphertext Attack

In this attack scenario, the attacker is assumed to be able to input several codewords with lengths satisfying (1), and obtain the corresponding symbol sequences. In other words, we assume that Assumption 1 stated in Section III is satisfied, which is a benefit to the attacker.

Although the adaptive chosen-ciphertext attack described in Section III is not directly applicable here, since the bit-wise permutation step is removed, we can still borrow some ideas from Section III to evaluate the security of the improved system. Let the input codeword be B. Then, the bit stream actually input to the ISAC decoder is $C = B \oplus \mathbf{E}$. Suppose now the attacker wishes to find which symbol is split in the first encoding step. A possible approach is to decode some equally spaced codewords, namely, $B_i = (f_i)_2^{N_c}$, where $f_i = (i-1)/P$, for $1 \le i \le P$, and P is the number of the sampled codewords. Let the decoded symbol sequences of B_i be S_i, and denote the first symbol of S_i be s_i^1. Without the XOR operation, it is easy to find out which symbol is split in the first encoding step by observing the s_i^1. The attacker can group all s_i^1 into a sequence $T = s_1^1 s_2^1 \cdots s_P^1$. If T exhibits the pattern $\mathbf{AA}\cdots\mathbf{A}\ \mathbf{BB}\cdots\mathbf{B}\ \mathbf{AA}\cdots\mathbf{A}$, the attacker can immediately determine that the symbol split in the first encoding step is $\mathbf{A}$. On the contrary, if T exhibits the pattern $\mathbf{BB}\cdots\mathbf{B}\ \mathbf{AA}\cdots\mathbf{A}\ \mathbf{BB}\cdots\mathbf{B}$, then the symbol split in the first encoding step is $\mathbf{B}$. Due to the XOR operation, however, this relationship becomes invalid. Even in the case that $\mathbf{A}$ is split in the first encoding step, the pattern $\mathbf{BB}\cdots\mathbf{B}\ \mathbf{AA}\cdots\mathbf{A}\ \mathbf{BB}\cdots\mathbf{B}$ can still be possible. Take the example that $p(\mathbf{A}) = 2/3$, $p(\mathbf{B}) = 1/3$, $N = 16$, and $k_1 = 3/16$. To the benefit of the attacker, we assume that the attacker can input short codewords, e.g., length-3 codewords. In Table II, we show the relationship between the key stream $\mathbf{E}$ and s_i^1, in the case that the input codeword length $N_c = 3$. It can be seen that when $\mathbf{E} = 000$ or $\mathbf{E} = 111$, the attacker will decide that the symbol split in the first encoding step is $\mathbf{A}$, while when $\mathbf{E} = 011$ or $\mathbf{E} = 100$, the attacker will decide that the symbol split in the first encoding step is $\mathbf{B}$. In the remaining four cases, the attacker cannot decide which symbol is split using the aforementioned method. Therefore, without knowing the key stream $\mathbf{E}$, the probability that $\mathbf{A}$ is split is equal to that of $\mathbf{B}$ is split. In other words, by using this method, the attacker still cannot reduce the uncertainty of which symbol is split in the first encoding step.

Another class of widely used chosen-ciphertext attack is to construct pairs of codewords with small differences, and try to find out some information about the secret key by comparing their outputs. This attack is usually powerful against those cryptosystems with poor diffusion property. When evaluating the security of the improved system, a natural way is to decode $B = b_1 b_2 \cdots b_{N_c-1} b_{N_c}$ and $B' = b_1 b_2 \cdots b_{N_c-1} \bar{b}_{N_c}$, where N_c satisfies (1). Denote the decoded symbol sequences of B and B' as S and S', respectively. The attacker can first

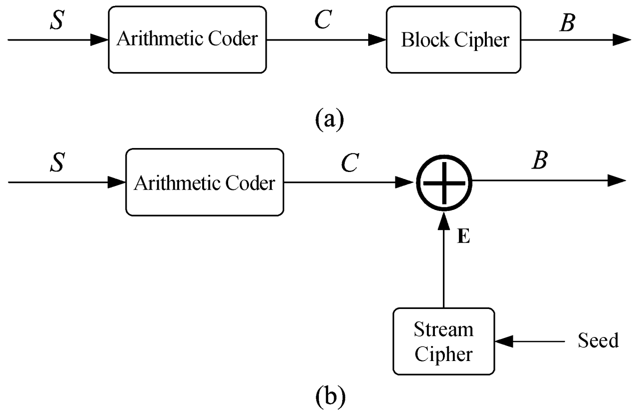

Fig. 10. Classical separate compression-encryption schemes. (a) AC concatenated with a block cipher; (b) AC concatenated with a stream cipher.

choose $N_c = N_{c,\max}$. Since $|(B)_d - (B')_d| = 2^{-N_{c,\max}}$, it is very likely that $S = S'$. Then, the attacker can gradually decrease N_c such that S and S' differ in the Nth symbol. Therefore, one of the split locations is within the interval $[\min\{(C)_d, (C')_d\}, \max\{(C)_d, (C')_d\})$, where $C = B \oplus \mathbf{E}$ and $C' = B' \oplus \mathbf{E}$. However, as $\mathbf{E}$ is unknown for the attacker, it is still difficult to find the split location.

D. Performance Analysis

Due to the elimination of the input symbol permutation step, the improved system can utilize the full-context information, which enables the context-based coding. This leads to the fact that the improved system may provide higher compression ratio, and could be more conveniently incorporated with the new generation of standards in which the context-based coding is employed.

In addition, in the original SAC, the output codeword permutation step cannot be performed until the whole bit stream from the ISAC encoder is generated. This introduces some amount of delay to the encoding/decoding. However, in the improved system, since we simply use the bit-wise XOR operation, the bit stream from the ISAC encoder can be immediately processed without the need of waiting for the whole bit stream.

E. Comparison With the Classical Separate Compression-Encryption Schemes

A classical approach to provide simultaneous compression and encryption is to concatenate a traditional arithmetic coder with a block cipher, e.g., AES, which is denoted by AC/AES system [1], as shown in Fig. 10(a). It was demonstrated in [1] that the number of transform steps is significantly larger in AES than that in the permutations of the SAC. This results in the fact that the throughput of the SAC could be higher than that of the AC/AES system. Since the permutations have been removed in the improved system, the throughput could be even higher. In addition, in the AC/AES system, the zero padding will cause some amount of coding efficiency loss, due to the block nature of AES. For both the SAC and the improved system, however, we can deal with arbitrary codeword length without any zero padding operations.

Another classical approach to provide simultaneous compression and encryption is to concatenate a traditional arithmetic coder with a stream cipher, e.g., RC4, which is denoted by AC/SC, as shown in Fig. 10(b). Since there is no private information in the arithmetic coder, the intermediate bit stream C in Fig. 10(b) could be easily obtained from the symbol sequence S. Then, the currently used key stream $\mathbf{E}$ could be easily recovered by $\mathbf{E} = B \oplus C$. Therefore, in order to achieve high level of security, the same key stream $\mathbf{E}$ can only be used just once. In practice, the initialization vector based technique could be applied to generate a unique key stream independent from other streams produced by the same seed, without having to go through a rekeying process. It should be pointed out that inappropriate integration with an initialization vector leads to severe security vulnerability, as reported in [26]. In the improved system, however, the intermediate bit stream C is unknown to the attacker who does not know the splitting key vector $\mathbf{K}$. The key stream used in the XOR operation of the improved system then cannot be easily derived in a similar way as done in the AC/SC system. This allows us to use the key stream $\mathbf{E}$ multiple times while not influencing the security of the system. Hence, in the improved system, we could avoid using the initialization vector based technique, which itself may result in security vulnerability [26]. In addition, in the AC/SC system, a secure pseudorandom number generator has to be used, making it computationally infeasible to infer the seed from its generated key stream. This requirement poses great challenges to the design of the pseudorandom number generator. Nevertheless, in the improved system, since the key stream $\mathbf{E}$ is unknown to the attacker, the requirement on the key scheduler could be much lower than that of a secure pseudorandom number generator. In fact, the simple keyed XOR operation proposed in [1] could be used to satisfy this purpose.

V. Conclusion

In this paper, we have addressed the security problem of the recently proposed encryption scheme secure arithmetic coding (SAC). We have suggested an adaptive chosen-ciphertext attack that can recover the key vectors used in the codeword permutation step with complexity $\mathcal{O}(N)$, where N is the symbol sequence length. This indicates that the SAC is not suitable for those applications where the attacker can have access to the decoder. Furthermore, we have discussed an improved version of the SAC such that it can resist the adaptive chosen-ciphertext attack and can be conveniently incorporated with the context-based coding.

Appendix
Proof of the Proposition 1

It can be easily found that it suffices to prove the lower bound in the case of $\lfloor\tau\rfloor \le Np(\mathbf{A})$, since any probability is no less than 0.

We have

$$\begin{aligned}\Pr[N(\mathbf{A}) > \lfloor\tau\rfloor] &= 1 - \Pr[N(\mathbf{A}) \le \lfloor\tau\rfloor] \\ &\ge 1 - \exp\left\{-\frac{1}{2p(\mathbf{A})}\frac{(p(\mathbf{A})N - \lfloor\tau\rfloor)^2}{N}\right\}\end{aligned} \quad (20)$$

where the last inequality holds from the fact that $\lfloor \tau \rfloor \leq Np(\mathbf{A})$ and the well-known Chernoff's bound [21]. It should be noted that the Chernoff's bound only holds when $\lfloor \tau \rfloor \leq Np(\mathbf{A})$. This completes the proof. □

Acknowledgment

The authors would like to thank the anonymous reviewers and the Associate Editor Prof. C. Guillemot for their very constructive and helpful comments. The authors would also like to thank Prof. P. Moulin of the University of Illinois at Urbana-Champaign for helpful discussions.

References

[1] H. Kim, J. T. Wen, and J. D. Villasenor, "Secure arithmetic coding," *IEEE Trans. Signal Process.*, vol. 55, no. 5, pp. 2263–2272, May 2007.
[2] C. Wu and C.-C. J. Kuo, "Design of integrated multimedia compression and encryption systems," *IEEE Trans. Multimedia*, vol. 7, pp. 828–839, Oct. 2005.
[3] M. Grangetto, E. Magli, and G. Olmo, "Multimedia selective encryption by means of randomized arithmetic coding," *IEEE Trans. Multimedia*, vol. 8, pp. 905–917, Oct. 2006.
[4] J. T. Wen, H. Kim, and J. D. Villasenor, "Binary arithmetic coding with key-based interval splitting," *IEEE Signal Process. Lett.*, vol. 13, pp. 69–72, Feb. 2006.
[5] R. Bose and S. Pathak, "A novel compression and encryption scheme using variable model arithmetic coding and couple chaotic system," *IEEE Trans. Circuits Syst. I*, vol. 53, pp. 848–857, Apr. 2006.
[6] Y. Mao and M. Wu, "A joint signal processing and cryptographic approach to multimedia encryption," *IEEE Trans. Image Process.*, vol. 15, pp. 2061–2075, Jul. 2006.
[7] 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.
[8] A. Skodras, C. Christopoulos, and T. Ebrahimi, "The JPEG 2000 still image compression standard," *IEEE Signal Process. Mag.*, vol. 18, pp. 36–58, Sep. 2001.
[9] T. Wiegand, G. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 560–576, Jul. 2003.
[10] I. H. Witten and J. G. Clearly, "On the privacy offered by adaptive text compression," *Comput. Secur.*, vol. 7, pp. 397–408, 1988.
[11] H. A. Bergen and J. M. Hogan, "Data security in a fixed-model arithmetic coding compression algorithm," *Comput. Secur.*, vol. 11, pp. 445–461, Sep. 1992.
[12] H. A. Bergen and J. M. Hogan, "A chosen plaintext attack on an adaptive arithmetic coding compression algorithm," *Comput. Secur.*, vol. 12, pp. 157–167, Mar. 1993.
[13] P. W. Moo and X. Wu, "Resynchronization properties of arithmetic coding," in *Proc. IEEE Int. Conf. Image Process.*, Oct. 1999, pp. 545–549.
[14] X. Wu and P. W. Moo, "Joint image/video compression and encryption via high-order conditional entropy coding of wavelet coefficients," in *Proc. IEEE Int. Conf. Multimedia Computer Syst.*, Jul. 1999, pp. 908–912.
[15] A. Barbir, "A methodology for performing secure data compression," in *Proc. 29th Southeast. Symp. System Theory*, Mar. 1997, pp. 266–270.
[16] X. Liu, P. Farrel, and C. Boyd, "A unified code," in *Proc. Int. Conf. Cryptography Coding*, Dec. 1999, vol. 1746, pp. 84–93.
[17] H. Ishibashi and K. Tanaka, "Data encryption scheme with extended arithmetic coding," in *Proc. SPIE*, Dec. 2001, vol. 4475, pp. 222–233.
[18] J. T. Zhou and O. C. Au, "Comments on "A novel compression and encryption scheme using variable model arithmetic coding and couple chaotic system"," *IEEE Trans. Circuits Syst. I*, vol. 55, pp. 3368–3369, Nov. 2008.
[19] B. Furht and D. Kirovski, *Multimedia Security Handbook*. Boca Raton, FL: CRC Press, Dec. 2004.
[20] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. Boca Raton, FL: CRC Press, 1997.
[21] R. Motwani and P. Raghavan, *Randomized Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 1995.
[22] J. Chou and K. Ramchandran, "Arithmetic coding-based continuous error detection for efficient ARQ-based image transmission," *IEEE J. Sel. Areas Commun.*, vol. 18, pp. 861–867, Jun. 2000.
[23] M. Grangetto, E. Magli, and G. Olmo, "A syntax-preserving error resilience tool for JPEG 2000 based on error correcting arithmetic coding," *IEEE Trans. Image Process.*, vol. 15, pp. 807–818, Apr. 2006.
[24] T. Guionnet and C. Guillemot, "Soft decoding and synchronization of arithmetic codes: Application to image transmission over noisy channels," *IEEE Trans. Image Process.*, vol. 12, pp. 1599–1609, Dec. 2003.
[25] P. G. Howard and J. S. Vitter, "Arithmetic coding for data compression," *Proc. IEEE*, vol. 82, pp. 857–865, Jun. 1994.
[26] S. R. Fluhrer, I. Mantin, and A. Shamir, "Weaknesses in the key scheduling algorithm of RC4," *Sel. Areas Cryptogr.*, pp. 1–24, 2001.

Jiantao Zhou (S'09) received the B.E degree from the Department of Electronic Engineering, Dalian University of Technology, Dalian, China, in 2002 and the Master's degree from the Department of Radio Engineering, Southeast University, Nanjing, China, in 2005. Currently, he is working towards the Ph.D. degree in the Department of Electronic and Computer Engineering, Hong Kong University of Science and Technology.

From September 2007 to August 2008, he visited the Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign under the support of Fulbright Hong Kong Dissertation Program. His research interests include multimedia encryption, information theory, chaotic cryptography, and neural cryptography.

Mr. Zhou was a coauthor of a paper that received the Best Paper award in IEEE Pacific-Rim Conference on Multimedia (PCM) in 2007.

Oscar C. Au (S'87–M'90–SM'01) received the B.A.Sc. degree from the University of Toronto, Toronto, ON, Canada, in 1986, and the M.A. and Ph.D. degrees from Princeton University, Princeton, NJ, in 1988 and 1991, respectively.

After being a Postdoctoral Researcher at Princeton University for one year, he joined the Department of Electrical and Electronic Engineering, Hong Kong University of Science and Technology (HKUST), in 1992. He is now an Associate Professor, Director of Multimedia Technology Research Center (MTrec), and Advisor of the Computer Engineering (CPEG) Program in HKUST. His main research contributions are on video and image coding and processing, watermarking and light weight encryption, speech and audio processing. Research topics include fast-motion estimation for MPEG-1/2/4, H.261/3/4 and AVS, optimal and fast suboptimal rate control, mode decision, transcoding, denoising, deinterlacing, post-processing, multiview coding, scalable video coding, distributed video coding, subpixel rendering, JPEG/JPEG2000 and halftone image data hiding. He has published about 200 technical journal and conference papers. His fast-motion estimation algorithms were accepted into the ISO/IEC 14496-7 MPEG-4 international video coding standard and the China AVS-M standard. He has three U.S. patents and is applying for over 40 more on his signal processing techniques. He has performed forensic investigation and stood as an expert witness in the Hong Kong courts many times.

Dr. Au has been an Associate Editor of the IEEE Transactions on Circuits and Systems Part I and the IEEE Transactions on Circuits and Systems for Video Technology. He is the Chairman of the Technical Committee (TC) on Multimedia Systems and Applications (MSATC) and a member of the TC on Video Signal Processing and Communications (VSPC) and the TC on DSP of the IEEE Circuits and Systems (CAS) Society. He served on the Steering Committee of IEEE Transactions on Multimedia and the IEEE International Conference on Multimedia and Expo (ICME). He also served on the organizing committee of the IEEE International Symposium on Circuits and Systems (ISCAS) in 1997, the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) in 2003, the ISO/IEC MPEG Seventy-First Meeting in 2004, International Conference on Image Processing (ICIP) in 2010, and other conferences.

Peter Hon-Wah Wong (M'01) received the B.Eng. degree (first-class hons.) in computer engineering from the City University of Hong Kong in 1996 and the M.Phil. and Ph.D. degrees in electrical and electronic engineering from the Hong Kong University of Science and Technology (HKUST) in 1998 and 2003, respectively.

He was a Postdoctoral Fellow at the Department of Information Engineering, Chinese University of Hong Kong (CUHK), from 2003 to 2005. He worked as at the Applied Science and Technology Research Institute Company Limited (ASTRI) as a Member of Professional Staff from 2005 to 2007. He was the Visiting Assistant Professor at the Department of Electronic and Computer Engineering, HKUST, from 2007 to 2008. He is currently the R&D Director of VP Dynamics Ltd., Hong Kong. His research interests include digital data hiding and watermarking, time scale modification, fast-motion estimation, video/image denoising, audio coding, audio enhancement, auto white balancing, high dynamic range image processing, and subpixel rendering.

Dr. Wong served on the organizing committee of the ISO/IEC MPEG Seventy-First Meeting in 2004, the International Symposium on Intelligent Signal Processing and Communications Systems (ISPACS) in 2005, and the Pacific-Rim Conference on Multimedia (PCM) in 2007. He received the Schmidt award of excellence in 1998. He is a member of Sigma Xi.