SECURITY AND COMMUNICATION NETWORKS
Security Comm. Networks 2016; **9**:43–53
Published online 6 October 2015 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/sec.1378

RESEARCH ARTICLE

Perturbation meets key-based interval splitting arithmetic coding: security enhancement and chaos generalization

Yushu Zhang[1], Di Xiao[2]*, Kwok-Wo Wong[3], Jiantao Zhou[4], Sen Bai[5] and Moting Su[6]

[1] School of Electronics and Information Engineering, Southwest University, Chongqing 400715, China
[2] College of Computer Science, Chongqing University, Chongqing 400044, China
[3] Department of Electronic Engineering, City University of Hong Kong, Kowloon, Hong Kong
[4] Department of Computer and Information Science, Faculty of Science and Technology, University of Macau, Taipa, Macau
[5] Department of Information Engineering, Chongqing Communication Institute, Chongqing 400035, China
[6] School of Economics and Business Administration, Chongqing University, Chongqing 400044, China

ABSTRACT

Key-based interval splitting arithmetic coding (KAC) possesses both encryption and compression capabilities. However, it possesses vulnerability to chosen-plaintext attack because the attacker can explore the relationship between the key and the codeword to deduce the secret key. In order to resist this attack, we propose to introduce perturbation into KAC. The perturbation-based KAC not only avoids the flaw of KAC that the splitting keys are usually located at the endpoint of certain codeword or at the border of two codewords but also removes the restriction that the keys are only allowed in certain sub-intervals, which result in great convenience to the key scheduler. In addition, based on generalized arithmetic coding using Generalized Luröth Series, we study the phase-space splitting of a chaotic map for generalized KAC and suggest the generalized perturbation-based KAC. This leads to the design of a joint compression and encryption scheme with more powerful cryptographic features. Copyright © 2015 John Wiley & Sons, Ltd.

KEYWORDS
arithmetic coding; perturbation; joint compression and encryption; chaotic map

***Correspondence**
Di Xiao, College of Computer Science, Chongqing University, Chongqing 400044, China.
E-mail: xiaodi_cqu@hotmail.com

1. INTRODUCTION

Arithmetic coding (AC) has been integrated in some coding standards such as JPEG2000 and H.264/AVC because of its high coding efficiency. Besides source coding, AC possesses certain secrecy properties for some specific applications. However, it is found insecure [1–3], and thereby, many variants have been proposed for the purpose of enhancing the security [4–9]. In particular, randomized arithmetic coding (RAC) [5] and key-based interval splitting arithmetic coding (KAC) [8] are two main schemes of embedding secure features in AC. RAC is based on the exchange of two encoding intervals in accordance with a key-generated secret sequence. A key bit is used for each source symbol to determine whether the binary fragments are exchanged or not [5]. Jakimoski and Subbalakshmi pointed out that RAC cannot outperform the standard one because of the generation requirement of one pseudo-random bit for each binary symbol [10]. Katti *et al.* demonstrated the insecurity of RAC on the assumption that the attacker cannot choose some chosen plaintexts [11].

In [12], Xiang *et al.* described a cryptanalysis of revealing the keystream used for exchanging decision. In addition, the interval swapping technique was also applied in a Slepian–Wolf code based on distributed arithmetic coding [13] and in a secure binary AC by means of digitalized modified logistic map and Linear feedback shift register (LFSR) [14]. Unlike RAC, KAC is a modification of AC in which each symbol may correspond to one contiguous interval or two non-adjacent intervals because of interval splitting [8]. It is not secure against chosen-plaintext attack described by Kim *et al.*, who further proposed the secure arithmetic coding (SAC) scheme that applied a string of permutations at the input and output siders [15]. The security of SAC upon adaptive chosen-ciphertext attack was studied in [16,17], followed by the security analysis of KAC under ciphertext-only attack with respect to message indistinguishability [18].

Chaotic system has the inherent characteristics such as pseudo-randomness and sensitivity to initial situations, which are related to the two fundamental attributes of a perfect cryptosystem: confusion and diffusion [19]. As a result, the amount of research work is for the ciphers based on chaos [20–29]. Interestingly, a viewpoint on chaotic cryptography was introduced in [30], in which one-time pad can be illustrated as the finding of the initializations of a couple of binary chaotic systems that are from a series of 1D nonlinear chaos equations referred to as Generalized Luröth Series (GLS). It is interesting to note that there exists a connection between chaos and AC, and GLS can be used to realize generalized arithmetic coding (GAC) [31–34]. In [32], Luca presented an interest compression scheme using a chaos-based symbol framework, that is, a source string matches with each trajectory in chaotic generator state space. Nagaraj further proved that AC is from GLS modes and GLS code can reach Shannon's entropy bound [33]. Besides, a compression-encryption scheme, where in order to attain scrambling effect, a secret key is used to govern the position and direction of the chaotic linear segments, is designed in [34]. A discrete piecewise linear system was employed by Lin *et al.* [31] to exploit GAC, which demonstrated that the compression capability can approximate the entropy bound.

Our contributions are twofold. On the one hand, in view of the vulnerability to chosen-plaintext attack of KAC, we propose to introduce perturbation into it. KAC is insecure under chosen-plaintext attack because the attacker can explore the relationship between the range of the key and the codeword to deduce the key information by using a sufficient number of segments with varying content and length. In order to break this relationship, a perturbation is introduced into KAC. When an interval is split into three subintervals by an inappropriate key, the right-most subinterval will be moved to the left of the left-most subinterval, and the two sub-intervals will merge into a new interval. This perturbation-based KAC (PKAC) makes the key be allowed in the full interval, which is very convenient to schedule the key. On the other hand, the generalization of PKAC by using chaos is also studied, which is meaningful because of the intrinsically cryptographic properties that chaos possesses. In line with the evolution from AC to GAC, we can obtain GKAC from KAC and GPKAC from PKAC.

The following are the abbreviations that appeared in the paper.

- AC: arithmetic coding.
- TAC: traditional arithmetic coding.
- RAC: randomized arithmetic coding.
- KAC: key-based interval splitting arithmetic coding.
- PKAC: perturbation-based interval splitting arithmetic coding with key.
- SAC: secure arithmetic coding.
- GAC: generalization of arithmetic coding.
- GRAC: generalization of randomized arithmetic coding.
- GKAC: generalization of key-based interval splitting arithmetic coding.
- GPKAC: generalization of perturbation-based interval splitting arithmetic coding with key.
- GLS: Generalized Luröth Series.

This paper is organized as follows. The KAC is reviewed in Section 2. Section 3 gives PKAC followed by its security analysis. Section 4 is devoted to the generalized PKAC. Section 5 discusses the compressibility and applications. Section 6 shows some conclusions and future work.

2. REVIEW OF KEY-BASED INTERVAL SPLITTING ARITHMETIC CODING

In traditional arithmetic coding (TAC), the interval connected with every symbol is contiguous. Nevertheless, in KAC, the interval connected with every symbol can be segmented to two subintervals with the help of a real number key known by both the encoder and decoder. we assume that $S = s_1 s_2 \cdots s_N$ represents the input string to be compressed, where $s_i \in \{A, B\}$ and N denote the number of source symbols. Assume that this discrete source follows the distribution $\{p(A), p(B)\}$ such that $p(A) + p(B) = 1$, where $p(A)$ denotes the probability of occurrence of the symbol A and $p(B)$ is the probability of occurrence of the symbol B. S is encoded by KAC using the key stream $K = k_1, k_2, \cdots, k_N$, with each element lying in the range $(0, 1)$.

As an example, suppose $p(A) = 2/3 = 0.67$ and $p(B) = 1/3 = 0.33$, and the first symbol $s_1 = A$. The initial half-open interval $[0, 1)$ is partitioned into two subintervals $I(A) = [0, 0.67)$ and $I(B) = [0.67, 1)$ proportional to $p(A)$ and $p(B)$, as shown in Figure 1(a). Here, $I(X)$ represents the interval for the symbol string X. If the key of KAC is $k_1 = 0.4$ within $I(A)$, the interval in relation to symbol A needs to be segmented. The splitting leads to that the proportion of $I(A)$ to the right of the key $k_1 = 0.4$ will be moved to the right of the $I(B)$. That is to say, $I(B)$ is shifted left to begin at k_1 (Figure 1(b) and (c)). After the splitting, $I(A) = [0, 0.4) \cup [0.73, 1)$ and $I(B) = [0.4, 0.73)$.

Prior to the second splitting, if the second symbol is $s_2 = A$, then one can partition $I(A)$ into $I(AA)$ and $I(AB)$ similar to TAC with the assumption that $I(A) = [0, 0.4) \cup [0.73, 1)$ is a contiguous interval, as indicated in Figure 1(d). Consequently, $I(AA) = [0, 0.4) \cup [0.73, 0.78)$ and $I(AB) = [0.78, 1)$. However, the splitting becomes constrained when multiple symbols are processed. Figure 1(e) explains the case of selecting an incorrect value for key k_2, which generates three subintervals corresponding to symbols AA. Figure 1(f) gives the correct range for k_2, which should lie only in the intervals I_1 and I_2 with length (I_1) = length (I_2). It should be noticed that, in Figure 1(f), length(I_1) = length(I_2) = length$(I(AB))$ if and only if length $(I(AB))$ < length(R), where $R = [0, 0.4)$ denotes the left-most interval in Figure 1(d). Otherwise, length$(I(AB))$ > length(R) and length(I_1) = length(I_2) = length$(I(R))$, as illustrated in Figure 1(g). The key k_2 is mapped to the interval I_1 or I_2. When $k_2 \in [0, 0.5)$, it is scaled to a value within I_1. On the contrary, $k_2 \in [0.5, 1)$ means that it corresponds to a value in I_2.

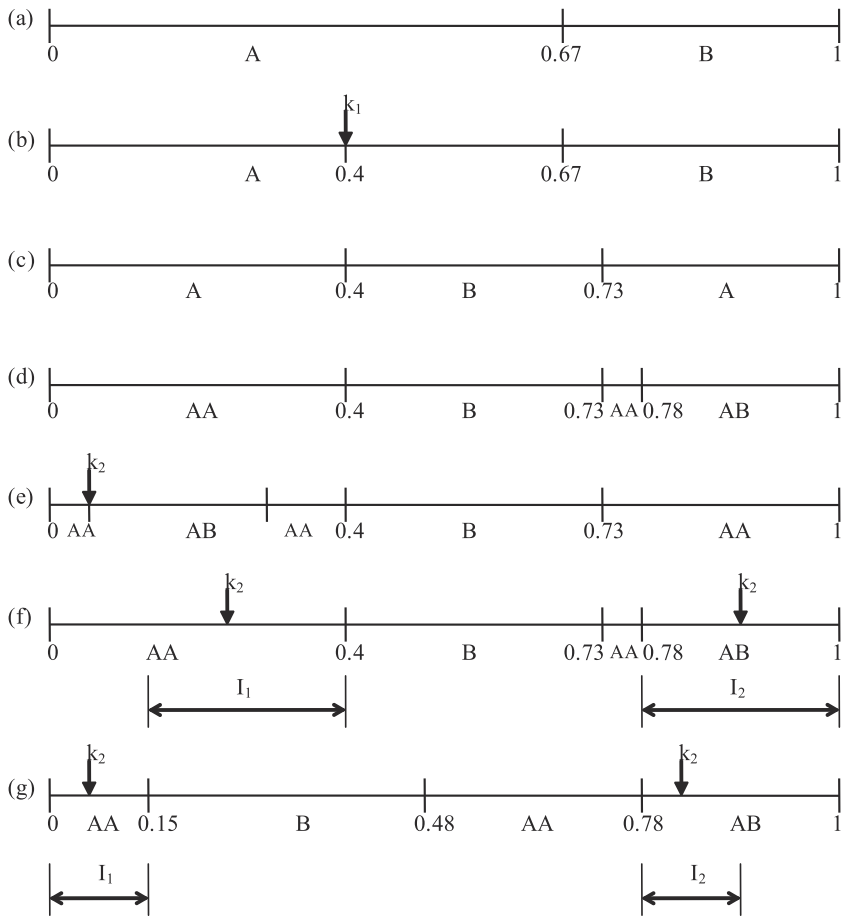

Figure 1. (a) Initial intervals $I(A)$ and $I(B)$. (b) Intervals $I(A)$ and $I(B)$ before splitting when $k_1 = 0.4$. (c) Intervals $I(A)$ and $I(B)$ after splitting when $k_1 = 0.4$. (d) Intervals $I(AA)$ and $I(AB)$ by partitioning interval $I(A)$ (without partitioning interval $I(B)$, similarly hereafter). (e) Intervals $I(AA)$ and $I(AB)$ after partitioning by using an inappropriate key. (f) Intervals I_1 and I_2 as the ranges of the appropriate value for the key k_2 when length($I(AB)$) < length(R). (g) Intervals I_1 and I_2 as the ranges of the appropriate value for the key k_2 when length ($I(AB)$) > length(R).

Suppose that the resulting final interval is $[start, end)$ after the whole input string S has been encoded, which is either one single contiguous interval or the union of at most two disjoint subintervals in $[0, 1)$. For the former case, the larger subinterval is chosen as the final interval. Any real number that is chosen in this interval could be selected as the codeword of S of length $\lceil -\log_2(end - start) \rceil$.

3. PERTURBATION-BASED KEY-BASED INTERVAL SPLITTING ARITHMETIC CODING

3.1. Perturbation

Key-based interval splitting arithmetic coding is insecure as the attacker can reveal the values of the splitting key vector K by contrasting codewords [15]. In light of the recursiveness of the encoding and splitting process, the attacker could alternatively provide input sequences of varying content and length by controlling the number of symbols and the specific strings so that the knowledge about K is gradually obtained. To fix this loophole, we suggest introducing perturbation into KAC. This can be exploited by moving the right-most subinterval to the left of the left-most subinterval only when an interval is split into three subintervals by an inappropriate key. The proposed KAC with perturbation (PKAC) offers another benefit that it removes the restriction that the splitting key is only allowed in the intervals I_1 and I_2 in the case of KAC. In other words, the key can span the full interval $(0, 1)$, which makes it convenient for key scheduling.

Here is an example of how perturbation is added into KAC. Assume the symbol string that needs to be compressed is $S = AA$ and $k_1 = 0.4$, then the splitting situation occurs in Figure 2(a). The $k_2 = 0.1$, and it is mapped to an absolute position in the interval $(0, 1)$. But it lies in neither I_1 nor I_2, and so, the interval $I(AA)$ is split into three subintervals as shown in Figure 2(b). Then, on the basis of PKAC, a new interval $I(AA)$ combining the left-most

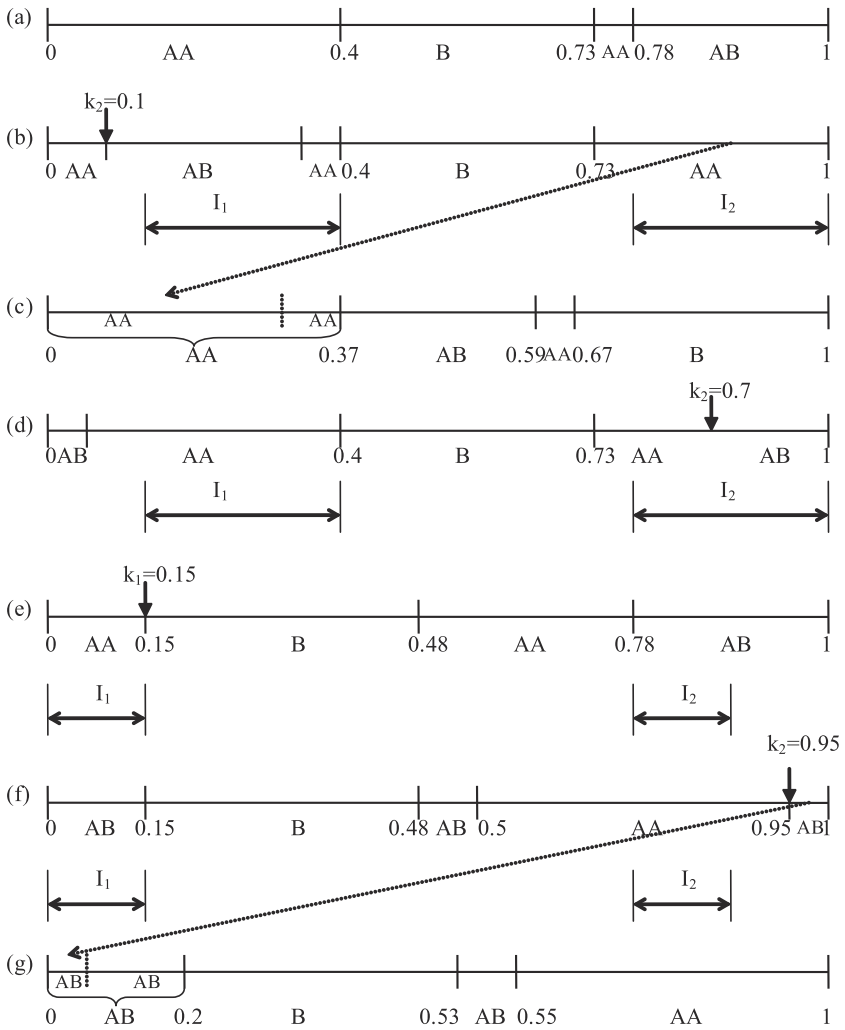

Figure 2. (a) Intervals $I(AA)$ and $I(AB)$ after splitting $I(A)$ by $k_1 = 0.4$. (b) Three subintervals of $I(AA)$ when $k_2 = 0.1$. (c) A new combined interval $I(AA) = [0, 0.37)$ after perturbation. (d) A situation of KAC when $k_2 = 0.7$. (e) Intervals $I(AA)$ and $I(AB)$ after splitting $I(A)$ by $k_1 = 0.15$. (f) Three sub-intervals of $I(AB)$ when $k_2 = 0.95$. (g) A new combined interval $I(AB) = [0, 0.2)$ after perturbation.

subinterval with the right-most subinterval is placed at the left-most part, and the remaining intervals will be shifted to the right accordingly, as shown in Figure 2(c). At last, $I(AA) = [0, 0.37) \cup [0.59, 0.67)$.

Note that the splitting point determined by the key is an absolute position. This encoding way of PKAC is more easily realized than that of KAC. In KAC, if $k_2 = 0.7, k_2 > 0.5$, one can map it to interval I_2 in the relative form rather than the whole interval $[0, 1)$ in the absolute form. In order to acquire the real k_2, I_2 is split into 2/5 because $0.7 - 0.5 = 0.2$, which is 2/5 of 0.5. The length of I_2 is equal to $1 - 0.78 = 0.22$, so k_2 is mapped to $0.78 + (2/5 \times 0.22) = 0.868$. Accordingly, $I(AA) = [0.088, 0.4) \cup [0.73, 0.868)$, as illustrated in Figure 2(d). This shows that the realization of the key in KAC is more cumbersome. In our scheme, the length of $I(AA)$ is always equal to that of KAC. This property implies that PKAC does not affect the coding efficiency.

Figures 2(e–g) illustrates another possible situation of PKAC corresponding to Figure 1(g). If $k_1 = 0.15, I(A) = [0, 0.15) \cup [0.48, 1)$ after the first splitting using k_1. It is further partitioned into $I(AA)$ and $I(AB)$ as shown in Figure 2(e). If $k_2 = 0.95$ belonging to neither I_1 nor I_2, it splits $I(AB)$ into three subintervals as drawn in Figure 2(f). The result of perturbation leads to $I(AB) = [0, 0.2) \cup [0.53, 0.55)$ as plotted in Figure 2(g). The interval length is equal to 0.22, that is, it remains unchanged.

Besides this perturbation way, other modes of perturbation are possible, as shown in Figure 3. With reference to the three emerging subintervals, each one can be relocated to the left or right of one of the other two subintervals, which results in four situations. As a result, there are 12 possible modes of perturbation. For example, in Figure 3(a), the third subinterval can be moved to four relocations: the left and right sides of the first one and that of the second one. The aforementioned perturbation

DOI: 10.1002/sec

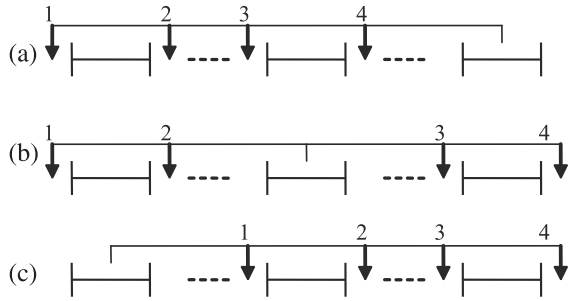

Figure 3. Four possible relocations when moving (a) the third subinterval, (b) the second subinterval, and (c) the first subinterval.

illustrated in Figures 1 and 2 corresponds to the situation depicted in Figure 3(a).

3.2. Security analysis

The relationship between the key and the codeword in KAC was discussed in [15]. It is assumed that the encoder always selects a position in the larger interval as the codeword when an input string is associated with two disjoint intervals due to interval splitting. It is also assumed that the encoder will stop the interval splitting after partitioning the interval corresponding to the last input symbol as there is no need for further encoding. In KAC, if the interval corresponding to symbol A is segmented and partitioned to generate two intervals $I(AA)$ and $I(AB)$, as illustrated in Figure 4, there are four possible length distributions of the locations of the codewords $C(AA)$ and $C(AB)$, where $C(X)$ represents the codeword of string X. The detailed explanations for these four cases are given as follow. SL denotes the splitting location in the absolute form. $L_l(X)$ and $L_r(X)$ indicate the length of the left and right intervals, respectively, when $I(X)$ comprises two distinct intervals $I_l(X)$ and $I_r(X)$.

If the interval corresponding to symbol B is split instead of A, similar analysis can be carried out. This is exactly the reason that the relationship between SL and the codeword exists. The attacker could launch a chosen-plaintext attack for KAC [15] by applying the following relationships. By encoding and comparing a sufficient number of input string consisting of two symbols, the adversary could gradually shrink the segment range and ultimately reveal the whole key.

Next, let us further investigate the connections between the key and the codeword in PKAC. The range of a key can be classified into two groups in a sense that whether a perturbation is appended. Figure 5 shows two different ranges for the four cases illustrated in Figure 4. For clarity, only the interval $I(A)$, including $I(AA)$ and $I(AB)$, is considered to be split. The ranges marked as R-I mean that the current PKAC without perturbation is the same as KAC, while the ranges marked as R-II imply that a perturbation will take effect. When the second splitting key k_2 is applied, either R-I or R-II is split, as depicted in Figure 5.

Figure 5 shows that, without perturbation, that is, in the case of KAC, both the keys k_1 and k_2 always lie in the boundary between two intervals as illustrated in R-I for each case. A worse case is that the keys are located at the endpoint of a codeword or at the border of two codewords so that they can be gradually revealed by comparing the codewords. However, after perturbation in PKAC, neither k_1 nor k_2 lies in the border of a codeword as indicated in each R-II, which efficiently hides the relationship between the key and the codeword. Only a perturbation is sufficient to hide the relationship to some extent. In general, encoding a sequence with many symbols results in many times of perturbation, which makes the attacker fail to deduce any

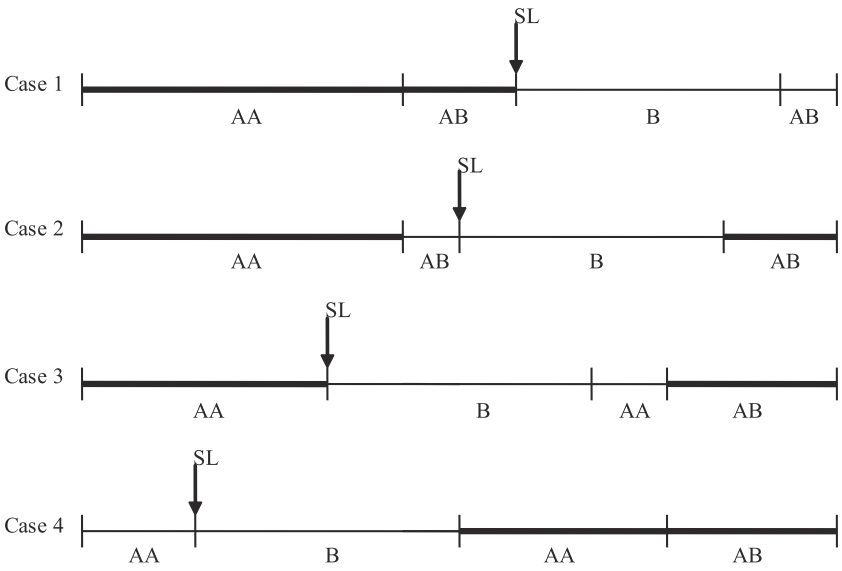

Figure 4. Four length distributions of the locations of the codewords $C(AA)$ and $C(AB)$.

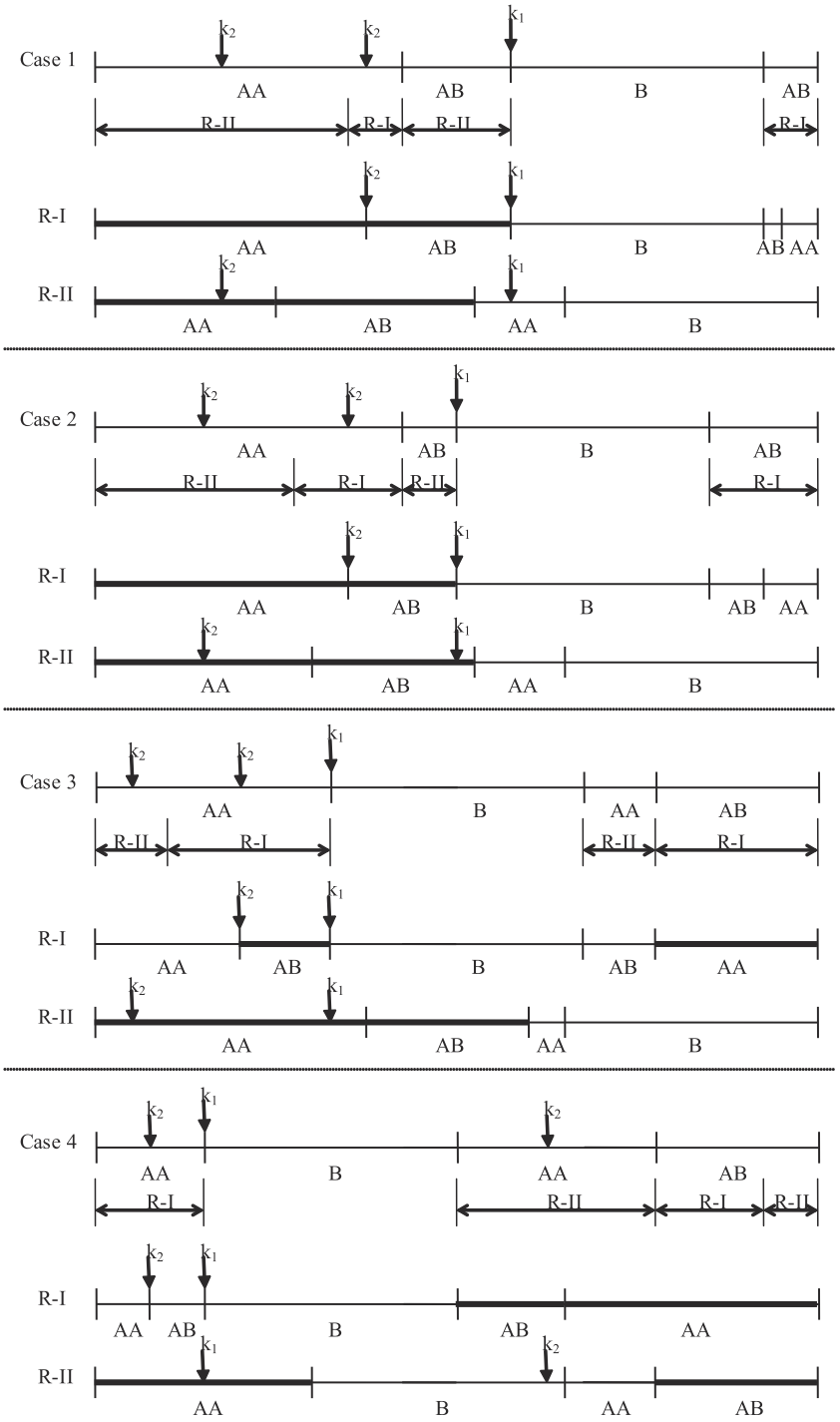

Figure 5. Two different ranges for the four cases of Figure 4 and the corresponding representative examples with different keys k_1 and k_2.

DOI: 10.1002/sec

useful information. Thus, the proposed PKAC is able to fix the security flaw of KAC.

In [15], Kim *et al.* introduced permutation into KAC to resist chosen-plaintext attack. This SAC scheme is composed of input permutation, KAC, and output permutation. Nonetheless, the security problems still exist [16–18]. The cause of the flaw is KAC itself rather than the permutations because they offer poor encryption. It is worth mentioning that the implementation complexity of the system based on permutation is not negligible. On the contrary, PKAC enhances the security of KAC without additional complexity. In terms of coding efficiency, SAC has the same performance as KAC because the permutations do not incur any compression efficiency penalty. Similarly, PKAC can maintain the original compression ratio of KAC. Considering the four cases in Figure 5, with respect to R-I without perturbation and R-II after perturbation, we summarize the relations and list them in Table I. The larger the codeword interval is, the higher the coding efficiency. Table I just gives some specific possible cases, and whether the codeword length of R-I is larger than that of R-II is indeterminate and random. This randomness is mainly introduced by the keys used in the interval splitting. However, it can be analyzed in a statistical way. If the keys are presented in a random order following the uniform distribution with zero expectation approximately, the number of cases that R-I > R-II is basically flat in comparison with that of R-II. As a consequence, the coding efficiency of PKAC is maintained.

Table I. Comparison of length between R-I and R-II for the length of codewords $C(AA)$ and $C(AB)$.

Four cases	Codeword length	Length comparison
Case 1	length($C(AA)$)	R-I > R-II
	length($C(AB)$)	R-I < R-II
Case 2	length($C(AA)$)	R-I > R-II
	length($C(AB)$)	R-I < R-II
Case 3	length($C(AA)$)	R-I < R-II
	length($C(AB)$)	R-I > R-II
Case 4	length($C(AA)$)	R-I > R-II
	length($C(AB)$)	R-I < R-II

4. GENERALIZATION

As mentioned earlier, a novel idea that GLS can be used to realize GAC was presented in [32,33]. It was found that there are eight modes of GLS and only one of them corresponds to binary AC. Without loss of generality, assume that the source generates two symbols A and B as presumed in the description of KAC with the probabilities of occurrence p and $1-p$, respectively. One of the GLS is a common piecewise linear chaos system given by

$$\text{Decoding:}\quad f(x)=\begin{cases} x/p, & 1\le x<p \\ (1-x)/(1-p), & p\le x<1 \end{cases}$$

and

$$\text{Encoding:}\quad f^{-1}(I)=\begin{cases} p\times I, & symbol=0 \\ p+(1-p)\times I, & symbol=1 \end{cases}$$

Figure 6 shows the eight modes of GLS followed by their mathematical representations. In fact, the last seven modes, that is, from Mode 2 to Mode 8, could be evolved from Mode 1. Consider the first four ones. In Mode 1, there exist two oblique lines corresponding to A and B, respectively, with positive slopes marked as "0". It is easy to come up with the modification of the slope direction that a positive slope can be changed to a negative one marked as "1". Altogether, four modes exist because of the two directions of each line as depicted from Mode 1 to Mode 4. Then, consider the remaining four modes. RAC refers to the exchange of the two encoding intervals at random according to a key-generated shuffling sequence [5].

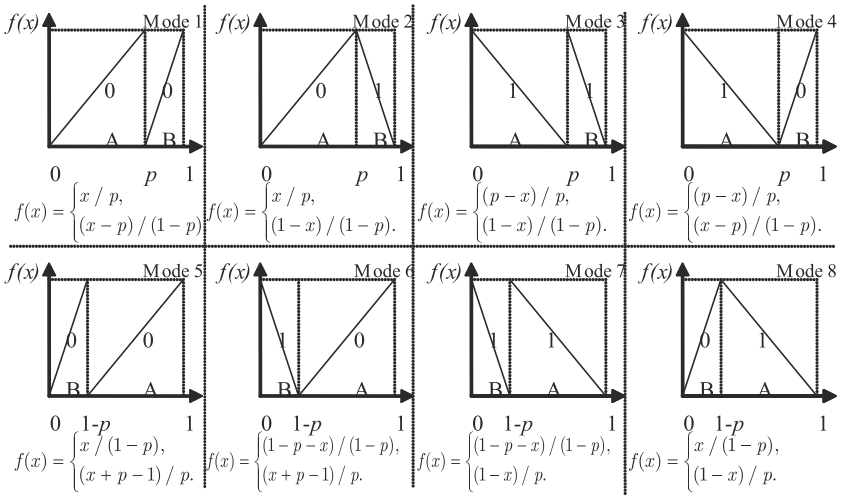

Figure 6. Eight modes of GLS and their mathematical representations.

Associating the interesting idea of interval swapping, we swap the two oblique lines in Modes 1–4 to output Modes 5–8 accordingly as the generalization of RAC (GRAC). As observed in the evolution process, these modes have the essence of RAC and have brought about some powerful cryptographic features into the TAC. As a consequence, it is well worth to work on the generalization of KAC and PKAC.

Before perturbation, GPKAC is the same as GKAC. First of all, consider how GKAC works with partitioning and splitting, whose process is shown in Figure 7(a–d). As for GAC with respect to binary source, the phase space $[0, 1)$ of a piecewise linear chaos map associated with the original interval $[0, 1)$ in TAC is divided into two parts proportional to $p(A)$ and $p(B)$, as depicted in Figure 6. In light of the way of interval splitting in KAC during the encoding process, we can infer the encoding process of GKAC. Take the previous example of $p(A) = 2/3 = 0.67$, $p(B) = 1/3 = 0.33$, and $k_1 = 0.4$. At first, the phase space $[0, 1)$ is divided into two parts denoted as $p(A)$ and $p(B)$, respectively, where $P(A) = [0, 0.67)$ and $P(B) = [0.67, 1)$. The slopes of the two oblique lines corresponding to $p(A)$ and $p(B)$ are $Slope(A) = 0$ and $Slope(B) = 1$, respectively, as shown in Figure 7(a). This form of the common piecewise

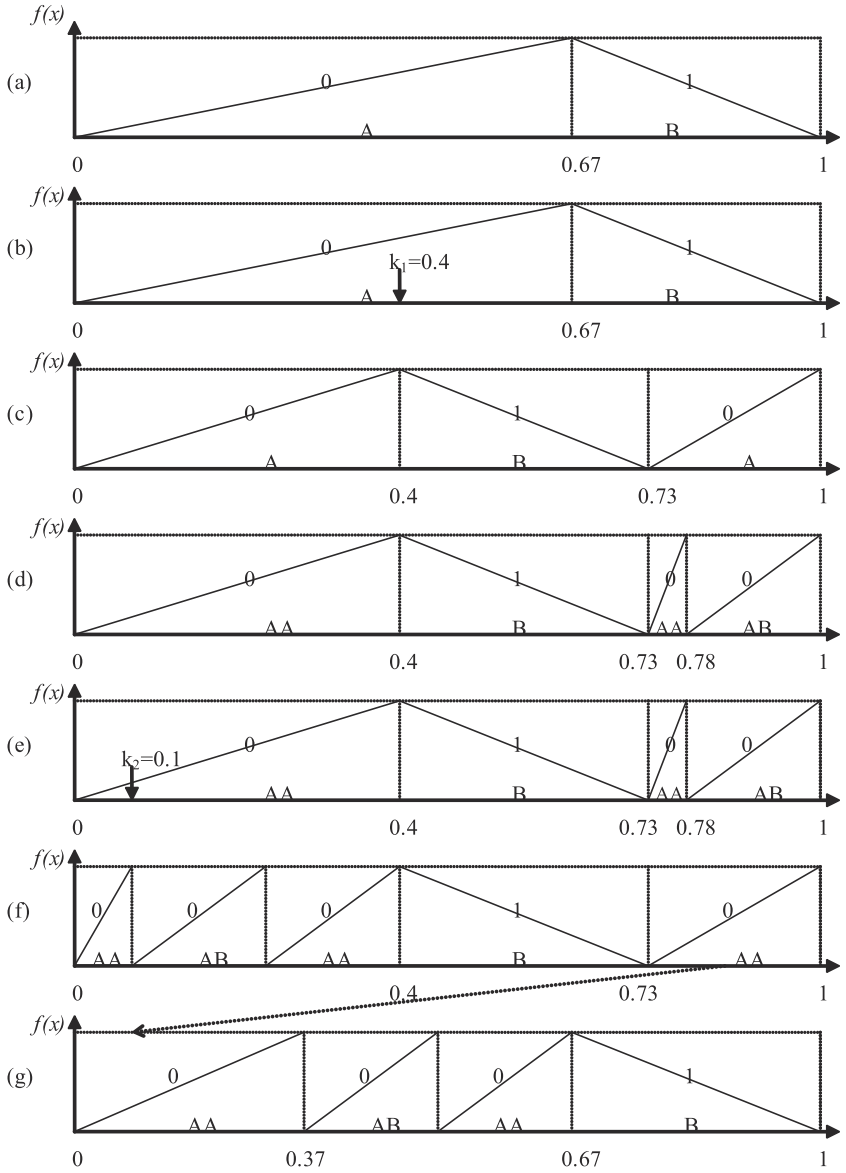

Figure 7. (a) Division of two phase subspaces $P(A)$ and $P(B)$ with respect to GKAC. (b) Two phase subspaces $P(A)$ and $P(B)$ before splitting when $k_1 = 0.4$. (c) Two phase subspaces $P(A)$ and $P(B)$ after splitting when $k_1 = 0.4$. (d) Division of two phase subspaces $P(AA)$ and $P(AB)$ with respect to $P(A)$. (e) Four phase subspaces before splitting when $k_2 = 0.1$. (f) Five phase subspaces after splitting when $k_2 = 0.1$. (g) A new subspace generated by merging the first $P(AA)$ and the last $P(AA)$.

linear chaos map is the same as Mode 2 in Figure 6. In Figure 7(b), the key $k_1 = 0.4$ means that the phase space $p(A)$ is split into two subspaces $P_l(A)$ and $P_r(A)$ representing the left and right ones, respectively. $P_r(A)$ will move to the right of $P(B)$ so that $P(A) = [0, 0.4) \cup [0.73, 1)$ and $P(B) = [0.4, 0.73)$ as shown in Figure 7(c). After splitting and moving, the slope with respect to $P(A_l)$ is no longer equal to that of $P(A_r)$ nor the original $P(A)$, that is, $Slope(A_l) \neq Slope(A_r) \neq Slope(A)$. Despite the variable size, the slope direction still remains unchanged. The second splitting of $P(A)$ results in $P(AA) = [0, 0.4) \cup [0.73, 0.78)$ and $P(AB) = [0.78, 1)$ (Figure 7(d)). Similar to KAC, the splitting key maps to the phase space with some constraints. Here, we do not repeat the description of the further splitting. In line with the evolution from KAC to GKAC, it must be capable of acquiring GPKAC from PKAC. The only thing to note is that after perturbation, two oblique lines will be merged into one, as shown in Figure 7(e–g).

Like Mode 2, the other seven modes of GLS can be split to achieve their corresponding generalizations. The main difference between KAC and GKAC (or GPKAC) is that in GKAC, every symbol X has two slopes $Slope(X) = 0$ and $Slope(X) = 1$ in association with the phase space $P(X)$, whereas there is no slope in KAC. Different slopes result in different encoding ways when some appropriate keys are exploited by selecting the slope direction. GKAC or GPKAC provides a mass of encoding structures, and thereby, it is a reasonable examinee for source coding encryption.

5. DISCUSSION

5.1. Compressibility

Table II shows the comparative results of coding efficiency between KAC and PKAC by encoding the input sequences with length $N = 10$, 100, and 1000 symbols. Two cases, that is, $p(A) = 0.60$ and $p(A) = 0.90$, are taken into consideration. In each case, 100 random sequences are generated and the average length of the corresponding 100 codewords is listed in Table II. The results confirm that the efficiency difference in percentage becomes smaller with the increase of N. When $N = 1000$, the difference falls to approximately 0.002% in both cases, the same as the theoretical analysis.

5.2. Applications

AC has been widely studied in many works [35–42] recently besides being adopted in the entropy coding stage of most international image and video coding standards. The variants of AC, including RAC, KAC, SAC and PKAC, can be embedded into some of these works to replace AC and applied to any multimedia coder standard employing AC to attain synchronous compression and encryption. However, RAC, KAC, and SAC have been proven insecure, and a number of permutation operations of SAC result in the increase of complexity. The proposed PKAC overcomes these drawbacks and can be extended to many practical applications. Take an example: it is well known that JPEG and JPEG2000 are often used for lossy or lossless compression of digital images. They offer progressive coding and region-of-interest coding. The compression ratio can be switched to provide an optional tradeoff between storage and quality. Their most significant feature is that they can reach excellent compression performance at the price of little image quality loss. Nevertheless, they do not provide any security protection. If some coding scheme possesses both excellent compression performance and sufficient level of security, its application scopes can be expanded. For this reason, the proposed PKAC technique can be used to modify the arithmetic coder, in JPEG or JPEG2000 to fill its gap. After the AC and its variants are generalized, the intrinsic properties of chaos can bring excellently cryptographic features. A representative example is the work in [43], in which GLS coding can be incorporated into JPEG without nearly affecting the original compression performance.

Table II. Comparative results of coding efficiency between KAC and PKAC.

Two probabilities	N	Entropy $\times N$	KAC	PKAC	Efficiency difference in %
$p(A) = 0.6$	10	9.71	11.58	11.67	7.770
$Entropy = 0.971$	100	97.1	98.77	98.72	–0.051
	1000	971	972.46	972.44	–0.002
$p(A) = 0.9$	10	4.69	6.13	6.01	–1.960
$Entropy = 0.469$	100	46.9	48.30	48.38	0.170
	1000	469	470.41	470.40	–0.002

KAC, key-based interval splitting arithmetic coding; PKAC, perturbation-based KAC.

Table III. Generalizations of AC and its variants.

AC [44,45]	RAC [5]	KAC [8]	**PKAC**(in this paper)
GAC [32,33]	GRAC [33]	**GKAC**(in this paper)	**GPKAC**(in this paper)

AC, arithmetic coding; RAC, randomized AC; KAC, key-based interval splitting AC; PKAC, perturbation-based KAC; GAC, generalized AC; GRAC, generalized RAC; GKAC, generalized KAC; GPKAC, generalized PKAC.

6. CONCLUSION AND PROSPECT

In order to present our idea clearly, we summarize the main conclusions in Table III, where AC and its three variants are listed in the first line, and the generalizations of these four coding schemes can be found in the second line. From Table III, two aspects are mainly studied in this paper. On the one hand, PKAC as a variant of KAC is proposed to improve the security of KAC. On the other hand, GKAC and GPKAC, corresponding to the generalized KAC and generalized PKAC, respectively, are introduced to supplement the generalizations.

There are two critical indicators including coding efficiency and precision to reflect the performance of a code. From the perspective of coding efficiency, RAC retains the same level as TAC. The compression ratio of KAC is slightly lower than that of AC, and the efficiency penalty becomes smaller with the increasing number of input symbols [8]. PKAC maintains a consistent compression ratio as KAC. The generalized forms of AC and its variants do not compromise the coding efficiency.

In practice, one can need sufficiently large integers rather floating point or real numbers as the endpoints of the current interval. Consequently, the integer form of a code is meaningful in the application. Integer AC and discrete GAC have been studied in [45] and [31], respectively. However, the other codes including RAC, KAC, PKAC, GRAC, GKAC, and GPKAC have never been investigated in terms of precision and coding speed, which are worth studying to further extend their applications.

ACKNOWLEDGEMENTS

The work was funded by the Chongqing Graduate Student Research Innovation Project (grant no. CYB14002), the Fundamental Research Funds for the Central Universities (grant nos. 106112013CDJZR180005, 106112014CDJZR 185501, and XDJK2015C077), the Natural Science Foundation of Chongqing Science and Technology Commission (grant nos. cstc2015jcyjA40039, cstc2012jjA40017, cstc 2013jcyjA40017, and cstc2013jjB40009), the National Natural Science Foundation of China (grant nos. 611731 78, 61272043, 61302161, 61472464, 61502399, 61402 547, and 61572089), Macau Science and Technology Development Fund (grant nos. FDCT/009/2013/A1 and FDCT/046/2014/A1), and Research Committee at University of Macau (grant nos. MRG007/ZJT/2015/FST, MRG 021/ZJT/2013/FST, MYRG2014-00031-FST, and MYRG 2015-00056-FST).

REFERENCES

1. Bergen HA, Hogan JM. Data security in a fixed-model arithmetic coding compression algorithm. *Computer Security* 1992; **11**(5): 445–461.
2. Bergen HA, Hogan JM. A chosen plaintext attack on an adaptive arithmetic coding compression algorithm. *Computer Security* 1993; **12**(2): 157–167.
3. Cleary JG, Irvine SA, Rinsma-Melchert I. On the insecurity of arithmetic coding. *Computer Security* 1995; **14**(2): 167–180.
4. Barbir A. A methodology for performing secure data compression. In *Proceedings of the Twenty-Ninth Southeastern Symposium on System Theory*, Cookeville, TN, 1997; 266–270.
5. Grangetto M, Magli E, Olmo G. Multimedia selective encryption by means of randomized arithmetic coding. *IEEE Transactions on Multimedia* 2006; **8**(5): 905–917.
6. Liu X, Farrell P, Boyd C. A unified code. In *Cryptogr. Cod.*, Lecture Notes in Computer Science. Springer: UK, 1999; 84–93.
7. Moo PW, Xiaolin W. Resynchronization properties of arithmetic coding. *Proceedings of the International Conference on Image Processing (ICIP)*, Snowbird, UT, 1999; 545–549.
8. Wen J, Kim H, Villasenor JD. Binary arithmetic coding with key-based interval splitting. *IEEE Signal Processing Letters* 2006; **13**(2): 69–72.
9. Xiaolin W, Moo PW. Joint image/video compression and encryption via high-order conditional entropy coding of wavelet coefficients. *IEEE International Conference on Multimedia Computing and Systems*, Florence, 1999; 908–912.
10. Jakimoski G, Subbalakshmi KP. Cryptanalysis of some multimedia encryption schemes. *IEEE Transactions on Multimedia* 2008; **10**(3): 330–338.
11. Katti RS, Srinivasan SK, Vosoughi A. On the security of randomized arithmetic codes against ciphertext-only attacks. *IEEE Transactions on Information Forensics and Security* 2011; **6**(1): 19–27.
12. Xiang T, Chenyun Y, Jinyu Q, Xinwen F. Cryptanalysis of secure arithmetic coding based on interval swapping. *8th International Conference & Expo on Emerging Technologies for a Smarter World (CEWIT)*, New York, 2011; 1–4.
13. Zhou J, Wong KW, Yang Y. Distributed arithmetic coding with interval swapping. *Signal Processing* 2015; **116**: 29–37.
14. Zhang Y, Di X, Wen W, Nan H, Moting S. Secure binary arithmetic coding based on digitalized modified logistic map and linear feedback shift register. *Communications in Nonlinear Science and Numerical Simulation* 2015; **27**(1-3): 22–29.
15. Kim H, Wen J, Villasenor JD. Secure arithmetic coding. *IEEE Transactions on Signal Processing* 2007; **55**(5): 2263–2272.
16. Sun HM, Wang KH, Ting WC. On the security of the secure arithmetic code. *IEEE Transactions on Information Forensics and Security* 2009; **4**(4): 781–789.
17. Zhou J, Oscar CA, Wong PHW. Adaptive chosen-ciphertext attack on secure arithmetic coding. *IEEE*

Transactions on Signal Processing 2009; **57**(5): 1825–1838.
18. Katti RS, Vosoughi A. On the security of key-based interval splitting arithmetic coding with respect to message indistinguishability. *IEEE Transactions on Information Forensics and Security* 2012; **7**(3): 895–903.
19. Shannon CE. Communication theory of secrecy systems. *Bell System Technical Journal* 1949; **28**(4): 656–715.
20. Chen J, Zhu Z, Chong F, Hai Y, Zhang Y. Reusing the permutation matrix dynamically for efficient image cryptographic algorithm. *Signal Processing* 2015; **111**: 294–307.
21. Hammami S. State feedback-based secure image cryptosystem using hyperchaotic synchronization. *ISA Transactions* 2015; **54**: 52–59.
22. He X, Li C, Huang T. Bogdanov–Takens singularity in tri-neuron network with time delay. *IEEE Transactions on Neural Networks and Learning Systems* 2013; **24**(6): 1001–1007.
23. Huang X, Ye G, Chai H, Xie O. Compression and encryption for remote sensing image using chaotic system. *Security in Communication Networks* 2015, DOI: 10.1002/sec.1289.
24. Liu H, Wang X. Color image encryption based on one-time keys and robust chaotic maps. *Computers & Mathematics with Applications* 2010; **59** (10): 3320–3327.
25. Sufi F, Han F, Khalil I, Jiankun H. A chaos-based encryption technique to protect ECG packets for time critical telecardiology applications. *Security in Communication Networks* 2011; **4**(5): 515–524.
26. Wong KW, Yuen CH. Embedding compression in chaos-based cryptography. *IEEE Transactions on Circuits and Systems II: Express Briefs* 2008; **55**(11): 1193–1197.
27. Di X, Liao X, Wong KW. Improving the security of a dynamic look-up table based chaotic cryptosystem. *IEEE Transactions on Circuits and Systems II: Express Briefs* 2006; **53**(6): 502–506.
28. Zhang Y, Di X, Shu Y, Li J. A novel image encryption scheme based on a linear hyperbolic chaotic system of partial differential equations. *Signal Processing-Image Communication* 2013; **28**(3): 292–300.
29. Zhou Y, Bao L, Chen CLP. A new 1D chaotic system for image encryption. *Signal Processing* 2014; **97**: 172–182.
30. Nagaraj N. One-time pad as a nonlinear dynamical system. *Communications in Nonlinear Science and Numerical Simulation* 2012; **17**(11): 4029–4036.
31. Lin Q, Wong KW, Chen J. Generalized arithmetic coding using discrete chaotic maps. *International Journal Of Bifurcation And Chaos* 2012; **22**(10): 1250256.
32. Luca MB, Serbanescu A, Azou S, Burel G. A new compression method using a chaotic symbolic approach. *Proceedings of IEEE Communication Conference*, Bucharest, Romania, 2004; 3–5.
33. Nagaraj N, Vaidya PG, Bhat KG. Arithmetic coding as a non-linear dynamical system. *Communications in Nonlinear Science and Numerical Simulation* 2009; **14**(4): 1013–1020.
34. Wong KW, Lin Q, Chen J. Simultaneous arithmetic coding and encryption using chaotic maps. *IEEE Transactions on Circuits and Systems II: Express Briefs* 2010; **57**(2): 146–150.
35. Heo J, Ho YS. Improved context-based adaptive binary arithmetic coding over H. 264/AVC for lossless depth map coding. *IEEE Signal Processing Letters* 2010; **17**(10): 835–838.
36. Weidong H, Wen J, Weiyi W, Han Y, Yang S, Villasenor J. Highly scalable parallel arithmetic coding on multi-core processors using LDPC codes. *IEEE Transactions on Communications* 2012; **60** (2): 289–294.
37. Kim SH, Ho YS. Fine granular scalable video coding using context-based binary arithmetic coding for bit-plane coding. *IEEE Transactions on Circuits and Systems for Video Technology* 2007; **17**(10): 1301–1310.
38. Liaw HT. The design of an arithmetic coding mechanism to determine relationships in a hierarchy. *Computers & Mathematics with Applications* 1999; **37**(9): 61–71.
39. Rhu M, Park IC. Optimization of arithmetic coding for JPEG2000. *IEEE Transactions on Circuits and Systems for Video Technology* 2010; **20**(3): 446–451.
40. Varma K, Damecharla HB, Bell AE, Carletta JE, Back GV. A fast JPEG2000 encoder that preserves coding efficiency: the split arithmetic encoder. *IEEE Transactions on Circuits and Systems I: Regular Papers* 2008; **55**(11): 3711–3722.
41. Wong KW, Lin Q, Chen J. Error detection in arithmetic coding with artificial markers. *Computers & Mathematics with Applications* 2011; **62**(1): 359–366.
42. Zhang L, Wang D, Zheng D. Segmentation of source symbols for adaptive arithmetic coding. *IEEE Transactions on Broadcasting* 2012; **58**(2): 228–235.
43. Zhang Y, Di X, Liu H, Nan H. GLS coding based security solution to JPEG with the structure of aggregated compression and encryption. *Communications in Nonlinear Science and Numerical Simulation* 2014; **19**(5): 1366–1374.
44. Rissanen J, Langdon GG. Arithmetic coding. *IBM Journal of Research and Development* 1979; **23**(2): 149–162.
45. Witten IH, Neal RM, Cleary JG. Arithmetic coding for data compression. *Communications of the ACM* 1987; **30**(6): 520–540.