# Using Chaotic Maps for Encrypting Image and Video Content

Amit Pande, Prasant Mohapatra
*Department of Computer Science*
*University of California Davis, USA*
*Email: {amit, prasant}@cs.ucdavis.edu*

Joseph Zambreno
*Electrical and Computer Engineering*
*Iowa State University, Ames, USA*
*Email: zambreno@iastate.edu*

***Abstract*—Arithmetic Coding (AC) is widely used for the entropy coding of text and multimedia data. It involves recursive partitioning of the range [0,1) in accordance with the relative probabilities of occurrence of the input symbols. In this paper, we present a data (image or video) encryption scheme based on arithmetic coding, which we refer to as Chaotic Arithmetic Coding (CAC). In CAC, a large number of chaotic maps can be used to perform coding, each achieving Shannon optimal compression performance. The exact choice of map is governed by a key. CAC has the effect of scrambling the intervals without making any changes to the width of interval in which the codeword must lie, thereby allowing encryption without sacrificing any coding efficiency. We next describe Binary CAC (BCAC) with some simple Security Enhancement (SE) modes which can alleviate the security of scheme against known cryptanalysis against AC-based encryption techniques. These modes, namely Plaintext Modulation (PM), Pair-Wise Independent Keys (PWIK), and Key and ciphertext Mixing (MIX) modes have insignificant computational overhead, while BCAC decoder has lower hardware requirements than BAC coder itself, making BCAC with SE as excellent choice for deployment in secure embedded multimedia systems. A bit sensitivity analysis for key and plaintext is presented along with experimental tests for compression performance.**

## I. Introduction

The issue of real-time multimedia delivery has gained an increased importance in a world dominated by portable multimedia-capable devices, as well as with the emergence of the cloud computing paradigm. The technical challenges involved in such scenarios include providing a delivery mechanism that is highly scalable, **secure**, easily search-able and index-able, all without losing the important **compression** properties. Providing security in a video communication context is especially challenging, as the security requirements tend to be application and platform-specific, and the input data is characterized by large storage requirements, real-time processing latencies, and the use of standardized video codecs.

Arithmetic coding is a data compression technique that encodes data by creating a code string which represents a fractional value on the interval $[0, 1)$. When a string is compressed using an arithmetic coder, frequently-used characters are stored with fewer bits and not-so-frequently occurring characters are stored with more bits, resulting in fewer bits used in total [1]. It typically enables very high coding efficiency as multiple symbols are coded jointly, and has been adopted for use in image compression standards, including JBIG-2, JPEG-LS, and JPEG2000, as well as video standards, including H.264/AVC, to provide lossless entropy coding.

Arithmetic coding is extremely efficient for compression efficiency in large data-sets and it achieves the Shannon compression efficiency for large chunks of data. However, as conventionally implemented, it is not particularly secure. A naive choice is to use well-known encryption methods such as the Advanced Encryption Standard (AES) in combination with a traditional arithmetic coder to satisfy both compression and security needs. However, this concept leads to increased computational complexity and the useful properties of compressed bitstream such as rate-adaptive transmission, scalability and DC-image extraction for content searching are lost [2]. These approaches encrypt the output of a compression system into cipher text, which is completely random and uncorrelated to the compressed bits. This makes it impossible to retain the desired properties of the compressed bitstream into the encrypted bitstream. The scheme presented in this paper overcomes these limitation because it doesn't modify any properties of the compressed bitstream.

Recently, Grangetto et al. [3] presented a Randomized Arithmetic Coding (RAC) scheme which achieves encryption by inserting some randomization in the arithmetic coding procedure at no expense in terms of coding efficiency. RAC needs a key of length 1-bit per encoded symbol. Wen et al. [4] presented a generalization of this procedure, referred to as Secure Arithmetic Coding (SAC). The SAC coder is constructed over a Key-Splitting Arithmetic Coding (KSAC) [5], where a key is used to split the intervals of an arithmetic coder and it adds input and output permutation to increase the security of the coder.

However, SAC introduces a loss in coding efficiency, particularly for small-sized inputs, which are later restricted to a small value by putting some constraints on the keyspace. The SAC encoder may have to work with multiple sub-intervals thereby significantly increasing the computational cost of encoding. Successful attacks have been demonstrated against this SAC scheme [6], [7], [8], [9].

We present a joint video encryption and compression

scheme based on piece-wise linear chaotic maps, referred to as Chaotic Arithmetic Coding (CAC). The idea of using chaotic maps for encryption was presented in [10]. However, the authors use a skew version for data encryption which is prone to known plaintext attacks, and increases the computational complexity of the coder exponentially.

The contributions of this paper are as follows:

1) We present a generalized framework for video encryption using chaotic maps called as Chaotic Arithmetic Coding (CAC). We discuss the general case with N alphabets and the specific case of binary alphabets.
2) The known weaknesses of arithmetic coding based encryption schemes are alleviated by our proposed security enhancements (SE).
3) CAC provides goals of encryption without any computational overhead. In fact, the decoding complexity of CAC is less than a normal Binary Arithmetic Coder (BAC) without encryption (1 multiplication and 1 addition per iteration vs. 1 multiplication and 2 addition per iteration for BAC).
4) CAC provides low-cost encryption without compression losses for multimedia data which is suitable for application in embedded systems scenarios where computational resources and power budget is limited.

## II. Arithmetic coding with Piece-wise Linear Chaotic Maps

Let us consider a scenario where we have a string $S = x_1, x_2, ...x_N$ consisting of $N$ symbols to be encoded. The probability of occurrence of a symbol $s_i$, $i \in 1, 2, ...n$ is given by $p_i$ such that $p_i = N_i/N$ and $N_i$ is the number of times the symbol $s_i$ appears in the given string $S$. We next consider a piece-wise linear map ($\rho$) with the following properties:

- It is defined on the interval $[0, 1)$ to $[0, 1)$ i.e.

$$\rho : \ [0, 1) \longrightarrow [0, 1)$$

- The map can be decomposed into N piece-wise linear parts $\varrho_k$ i.e.

$$\rho = \bigcup_{k=1}^{N} \varrho_k$$

- Each part $\varrho_k$ maps the region on the x axis $[beg_k, end_k)$ to the interval $[0, 1)$ i.e.

$$\varrho_k : \ [beg_k, end_k) \longrightarrow [0, 1]$$

The last two propositions lead to:

$$\bigcup_{k=1}^{N} [beg_k, end_k) = [0, 1)$$

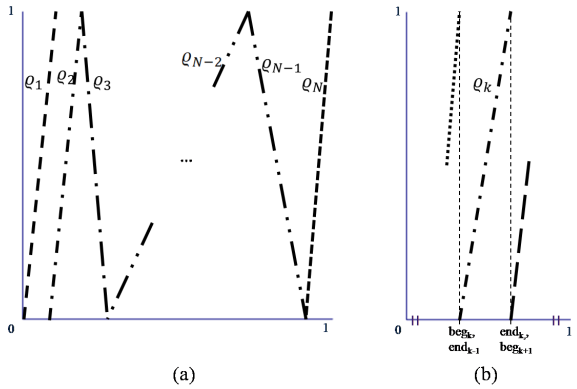


Figure 1. A sample piece-wise linear map for arithmetic coding like compression (a) The entire map is shown ($\rho$) (b) A single linear part of the map ($\varrho_k$) is zoomed. It can have a positive or negative slope depending on choice

- The map $\varrho_k$ is one-one and onto i.e.:

$$\forall x \in [beg_k, end_k)$$
$$\exists y \in [0, 1) : y = \varrho_k(x), \text{ and}$$
$$\forall y \in [0, 1)$$
$$\exists x \in [beg_k, end_k) : \varrho_k(x) = y$$

- $\rho$ is a many-one mapping from $[0, 1)$ to $[0, 1)$. This implies that the decomposed linear maps ($\varrho_k$) don't intersect each other i.e.

$$\forall (k \neq j) : [beg_k, end_k) \bigcap [beg_j, end_j) = 0$$

- Each linear map $\varrho_k$ is associated uniquely with one symbol $s_i$. The mapping $\varrho_k \longrightarrow s_i$ is defined arbitrarily but the one-one relationship must hold.
- The valid-input width of each map ($\varrho_k$), given by $(end_k - beg_k)$ is proportional to a probability of occurrence of symbol $s_i$.

$$end_k - beg_k \propto p_i$$
$$\Rightarrow end_k - beg_k = C \times p_i$$

We recall that $\sum_{k=1}^{N}(end_k - beg_k)$ is same as the input width of $\bigcup_{k=1}^{N} \varrho_k = \rho$, which is 1. Also, $\sum_{i=1}^{N} p_i = 1$. Thus, we get the value of constant $C$ to be 1.

$$\Rightarrow end_k - beg_k = p_i$$

Figure 1 shows a sample map fulfilling these properties. Figure 1(a) shows the full map with different parts $\varrho_1, \varrho_2, ...\varrho_N$ present, while Figure 1(b) zooms into individual linear part $\varrho_k$. The maps are placed adjacent to each other so that each input point is mapped into an output point in the range $[0, 1)$. The total number of distinct ways of arranging $N$ maps to obtain $\rho$ fulfilling the properties mentioned above is given by $N! = N \times (N-1) \times (N-2)...3 \times 2 \times 1$, where ! denotes the factorial sign. It is same as arranging these $N$ maps in

a sequence, one after another, with the end interval of one map touching the beginning interval of another.

However, there are $N$ different piece-wise maps, each with two possible orientations (with positive or negative slope). Thus, the number of total permutations possible is given by $N!2^N$ which is independent of unique symbol probability. Thus, for $N$-ary arithmetic coding or arithmetic coding with $N$ symbols, it is possible to have $N!2^N$ different mappings each leading to same compression efficiency. Since we can arbitrarily choose any 1 of the $N!2^N$ maps, the key space for encoding a single bit of data is $\lceil \log_2(N!2^N) \rceil$ bits, where $\lceil\rceil$ represents the greatest integer function. For $N = 2$, it gives 8 mappings. If we increase $N$ to 4 this value increases to 384.

The equation for individual maps can be derived as follows:

$$y' = \varrho_k(x') = \left(\frac{x' - beg_k}{end_k - beg_k}\right) \text{ or } \left(1 - \frac{x' - beg_k}{end_k - beg_k}\right)$$

The equation for the full map is given by

$$y = \rho(x) = \varrho_k(x) : beg_k \leq x < end_k$$

The coding procedure, correspondence to arithmetic coding and compression efficiency of basic chaotic coding is explained in [11].

### A. Compression Efficiency

The compression efficiency of the procedure lies in the width of the final interval from which we need to choose the initial value from. Let us consider encoding a general sequence of $N$ symbols such that probabilities of occurrence of the $i^{th}$ symbol is given by $\frac{N_i}{N}$ where $N_i$ is the number of occurrence of the symbol in the sequence. On every iteration, to encode an arbitrary symbol $N_j$, the width of interval (originally [0,1) and length 1) shrinks by a factor of $end_j - beg_j$ (width of $\varrho_j$). Thus, the width $\delta$ of final interval would be given by:

$$\delta = \prod_{j=1}^{N} \left(\frac{N_j}{N}\right)^{N_j}$$

The average number of bits required per symbol ($B_{av}$) is given by

$$B_{av} = \frac{B}{N} = \frac{1}{N}\left[-\sum_{j=1}^{N}(N_j)\log_2\left(\frac{N_j}{N}\right)\right]$$

This relation has been derived in [10], by averaging the total number of bits which is given by the logarithm of $\delta$. According to Shannon's entropy equation, the number of bits needed to encode a string of symbols is given by

$$B_{sh} = -\sum_{j=1}^{N}\frac{N_i}{N}\log_2\frac{N_i}{N}$$

$$B_{av} = \frac{1}{N}\lceil N \times B_{sh}\rceil \leq \frac{1}{N}(N \times B_{sh} + 1)$$

$$\Rightarrow B_{av} \leq B_{sh} + \frac{1}{N}$$

As $N \to \infty$, $B_{av} \to B_{sh}$. Thus, the proposed scheme gives optimal compression for large codewords.

### B. Binary Chaotic Arithmetic Coding (BCAC)

AC is more commonly implemented in binary mode to reduce the computational requirements of video coders. The Binary CAC (or BCAC) uses either of the eight equivalent skewed binary maps (shown in Figure 2) based on an input key. These maps differ from each other in the way input is mapped into the chaotic orbit - differ in the interval in which the arithmetic code must lie for a symbol '0' or '1' but the width of interval remains the same.

We define the generalized skewed binary map with the following equations:

$$y = \left\{ \begin{array}{l} n_1 x + c_1 \text{ when } x \leq k \\ n_2 x + c_2 \text{ when } x > k \end{array} \right\} \quad (1)$$

$$\text{Decode} \left\{ \begin{array}{l} \text{'0' when } x \in [i1, i2] \\ \text{'1' when } x \in [i3, i4] \end{array} \right\} \quad (2)$$

Then, the back iteration on skewed binary map is defined by the following equations:

$$x = \left\{ \begin{array}{l} m_1 y + b_1 \text{ when '0'} \\ m_2 y + b_2 \text{ when '1'} \end{array} \right\} \quad (3)$$

where $n_1$, $n_2$, $c_1$, $c_2$, $m_1$, $m_2$, $b_1$, and $b_2$ values can be precomputed for different maps and stored in a table for look-up for fast access. Table I gives the value of these parameters for all eight chaotic maps.

Grangetto et al. [3] present a Randomized Binary Arithmetic Coding (RBAC) scheme where they change the ordering of '0' and '1' intervals in a Binary Arithmetic Coder (BAC) based on a key. RBAC can be seen as a special case of BCAC where only two of the eight modes of BCAC are used for encryption purposes (drawn in Figure 2(a) and (e)). Similarly, KSAC [5] can be represented in terms of piecewise linear maps by removing the condition of continuity of individual maps ($\varrho_i(x)$). Each part $\varrho_i$ maps a discontinuous interval on x-axis to the interval [0,1).

#### Implementation Efficiency

For a normal binary arithmetic coder, at each iteration the starting interval $[I_s, I_e)$ is updated at one end. On encoding a '0' the final interval becomes $[I_s + p(I_e - I_s), I_e)$ while on encoding a '1' the final interval becomes $[I_s, I_s + p(I_e - I_s))$. Thus, every iteration requires one multiplication and two addition operations. The decoding procedure for a binary arithmetic coder involves updating the interval $[I_s, I_e)$ at one end depending on whether the last decoded symbol was a '0' or a '1'. Thus, every iteration again requires one multiplication and two addition operations.

Table I
PARAMETER LIST FOR THE EIGHT POSSIBLE CHOICES OF CHAOTIC ENCODER

| Parameter | (a) | (b) | (c) | (d) | (e) | (f) | (g) | (h) |
|---|---|---|---|---|---|---|---|---|
| M1 | $p$ | $p$ | $-p$ | $-p$ | $p$ | $-p$ | $-p$ | $p$ |
| B1 | 0 | 0 | $p$ | $p$ | $1-p$ | 1 | 1 | $1-p$ |
| M2 | $1-p$ | $p-1$ | $p-1$ | $1-p$ | $1-p$ | $1-p$ | $p-1$ | $p-1$ |
| B2 | $p$ | 1 | 1 | $p$ | 0 | 0 | $1-p$ | $1-p$ |
| N1 | $1/p$ | $1/p$ | $-1/p$ | $-1/p$ | $1/(1-p)$ | $1/(1-p)$ | $-1/(1-p)$ | $-1/(1-p)$ |
| C1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| N2 | $1/(1-p)$ | $-1/(1-p)$ | $-1/(1-p)$ | $1/(1-p)$ | $1/p$ | $-1/p$ | $-1/p$ | $1/p$ |
| C2 | $-p/(1-p)$ | $1/(1-p)$ | $1/(1-p)$ | $-p/(1-p)$ | $(p-1)/p$ | $1/p$ | $1/p$ | $(p-1)/p$ |
| I1 | 0 | 0 | 0 | 0 | $(1-p)$ | $(1-p)$ | $(1-p)$ | $(1-p)$ |
| I2 | $p$ | $p$ | $p$ | $p$ | 1 | 1 | 1 | 1 |
| I3 | $p$ | $p$ | $p$ | $p$ | 0 | 0 | 0 | 0 |
| I4 | 1 | 1 | 1 | 1 | $1-p$ | $1-p$ | $1-p$ | $1-p$ |
| K | $p$ | $p$ | $p$ | $p$ | $1-p$ | $1-p$ | $1-p$ | $1-p$ |

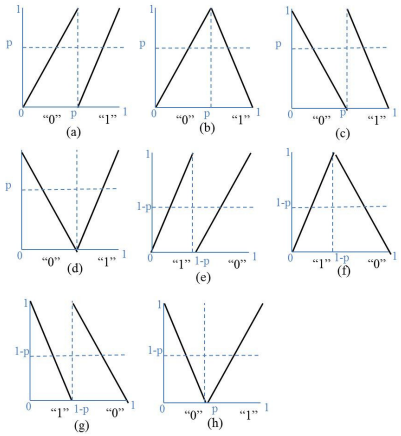

Figure 2. (a-h) show the eight modes of the skewed binary map ($p = 0.6$)

For BCAC, both ends of interval are updated at every iteration using a linear transformation $x = my+b$ thus requiring two multiplications and two additions for encoding. The decoding is simple as it involves iteration on the chaotic map according to the linear transformation $y = nx + c$ involving a multiplication and an addition operation. There are some additional table lookups (an 8-input LUT required for BCAC to choose the exact chaotic map) involved in chaotic coding to choose the right chaotic map at every iteration, which can be efficiently implemented in software or hardware. Thus, CAC encode requires more computations than BAC encode, while CAC decode requires less computations than BAC decode.

## III. SECURITY

### A. Application to Multimedia / Data Encryption

CAC is Shannon-optimal in terms of compression efficiency. By varying the mapping $\varrho_k \longrightarrow s_i$, we can obtain different maps, all of which give the same compression efficiency with different intervals for the final codeword. This parameterization of chaotic piece-wise maps allows us to build a keyspace for data/ video encryption using chaotic arithmetic coding. The choice of mapping is thus governed by an encryption key.

For BCAC, we have 8 possible maps for every encoded bit (see Figure 2) giving us up to 3 bits of encryption key per encoded symbol. The large keyspace makes the scheme secure against exhaustive trials. For full encryption, the entire volume of multimedia data is passed through the CAC encoder, while in case of selective encryption, only the important parts of data are passed through CAC encoder. We have a keyspace of $3N$ bits for $N$ bit plaintext. The large keyspace makes it extremely difficult to launch brute-force attacks. Since the key remains the same for multiple iterations, in effect the effective length of key bits is much less than the plaintext bits.

If we reveal the first K bits of the key publicly, then a part of the bitstream can be decoded correctly while decoding the entire bitstream will require knowledge of the entire key. In that case, CAC can be used to provide conditional access to part of multimedia content or scalable video encryption [12]. Scalable multimedia encryption finds its applications in modern pervasive / cloud-based multimedia applications where different types of users want to access the same multimedia content at different resolutions and access-privileges.

### B. Threat Model

BCAC coder / decoder pair is treated as encryption / decryption oracle, respectively. In our threat model, an attacker is able to choose a plaintext of chosen length and obtain its corresponding ciphertext from the encryption oracle. He can repeat this process for at most P times. In other words, we allow the attacker to adaptively select plaintext and use the encryption / decryption oracle for a polynomial number of times.

### C. Security Enhancements (SE)

As mentioned previously (in Section II), arithmetic coding-based encryption schemes (such as RAC and SAC) have been found to be vulnerable to cryptographic attacks (chosen ciphertext attack [7], ciphertext only attack and chosen plaintext attack [9]).

**Algorithm 1** PM mode

$\{PT(n)\}$ : Input to CAC encoder for (n)th pass
$\{CT(n)\}$ : Encoded Output of CAC Encoder for (n)th pass
$\{K(n)\}$ : Key value for (n)th pass
$\{I(n)\}$ : Plaintext Input for (n)th pass
$\{O(n)\}$ : Ciphertext Output for (n)th pass
$\{K_{3N}\}$ : Initiating Key
$\{W\}$ : PM parameter
**PM_mode()**
$K(n) = K_{3N}$
**for** $w = 1; j \leq W; j++$ **do**
    $PT(w) = I(w) \oplus K(w)$
**end for**
$O = CT$ =CAC.encode($PT$,$K$)

An attacker can guess the key, in $O(N)$ operations by giving different known inputs to the system (known-plaintext attack). In this section, we mention three SE modes for BCAC, which add considerable levels of security to the design. In this discussion, we use the following conventions: the encryption oracle has a BCAC encoder and some added operators for SEs. $K_{3N}$ is the initiating key value for the encryption oracle. $I(n)$ and $O(n)$ are the input and output sequences of encryption oracle for $n^{th}$ iteration. $PT(n)$, $CT(n)$ and $K(n)$ are the Plaintext, Ciphertext and Key values for the $n^{th}$ iteration of BCAC coder. The length of $PT(n)$ and $I(n)$ is $N$ bits, the compressed outputs $CT(n)$ and $O(n)$ is $M(n)$ bits ($M(n) \leq N$ in general), $K_{3N}$ and $K(n)$ is $3N$ bits.

*1) Plaintext Modulation (PM) Mode:* In the Plaintext modulation mode, the first $F$ bits of input plaintext are XORed with the input key values. The value of $F$ is chosen according to application requirements. When $F << N$ ($F$ is small), there is negligible losses in compression performance but the security level obtained is low. On the other side, when $F \approx N$, the security level is highest but the compression performance will be compromised. Thus, it is appropriate to choose large values of $N$ (say $N = 1000$) and relatively small $F$ ($F \approx 30$). The stepwise details of BCAC+PM algorithm is given in Algorithm 1 and shown in Figure 3(b). The idea of XORing the first few bits of plaintext with a key make it behave like a one-time pad at the beginning of BCAC. Moo and Wu [13] discuss how it is extremely difficult (exponential in $F$) to reconstruct the remaining plaintext in the arithmetic coding case, we lose the first $F$ bits. However, in the case of BCAC encoding, this complexity compounds manifold because of the large keyspace.

*2) Key and Output Mixing (MIX) Mode:* In the MIX mode, $CT(n)$ and $K(n)$ are mixed with each other to obtain $O(n)$ and $K(n+1)$. This operation is performed as follows:

1) We XOR $K(n)$ and $O(n)$ to obtain $K(n+1)$. Since $O(n)$ is of length $M(n) \leq 3N$, we cyclically repeat $O(n)$ to make it $3N$ bits long.
2) We rotate $O(n)$ with $L$ arbitrary bits to the left (cyclical left shift). This operation is easily performed in hardware using wire permutations, and in software using simple command for left rotate.
3) We XOR rotated $O(n)$ with first $M(n)$ bits of $K(n+1)$ to get the ciphertext output $O(n)$ of the encryption oracle.

All XOR operations can be implemented cheaply in commercial hardware and software. See Algorithm 2 and Figure 3(c) for description and figure. There is no loss in compression efficiency or throughput of the system.

**Algorithm 2** Mix mode

**Mix_mode()**
$\{A(n) \oslash B(n)\}$ : XOR A with cyclically extended or shrunk B, i.e. $A(n) \oplus B(n$ modulo(size(B)) for $1 \leq n \leq$ size(A)
$\{ROL(a,b)\}$ : Rotate b to Left by a bits
$\{L\}$: Number of bits to be rotated to left
$PT(n) = I(n)$
$CT(n)$ =CAC.encode($PT(n), K(n)$)
$K_{n+1}$=$Kn \oslash O(n)$
$O(n) = ROL(L, O(n)) \oslash K_{n+1}$

**Algorithm 3** PWIK mode

**PWIK_mode( )**
$\{PR\}$ : Largest prime in GF($2^{256}$)
$K(0) = K_{3N}$;
$PT(n) = I(n)$
$K(p)$=($K(p-1)$+InitValue$_2$)mod $2^{256}$
**if** ($K(p) <$ InitValue$_2$) **then**
    **return** $K(p)$=$K(p)$+$2^{256}$-$PR$
**end if**
$CT(n)$=CAC.Encode($PT(n), K(n)$)
$O(n) = CT(n)$

Mix mode allows efficient mixing of key and ciphertext, making it unintelligible for an attacker to recover the relationship between input and key values using output values.

*3) Pair Wise Independent Keys ($PWIK$) Mode:* In $PWIK$ mode, independent keys are generated for each

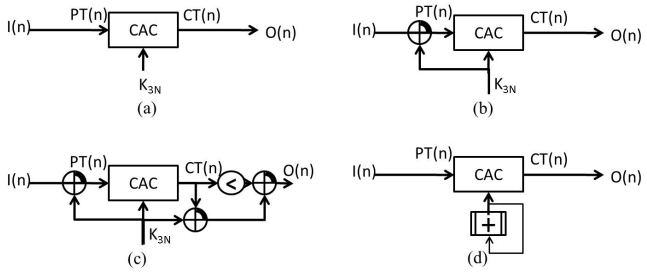


Figure 3. Different modes for SEs in CAC (a) Normal CAC encoder, (b) PM mode - The $N$ plaintext bits are mixed with the key, (c) MIX mode - $3N$ key bits are mixed with compressed output, the compressed output is rotated left by $L$ arbitrary bits and mixed with $M$ bits of mixed key, (d) PWIK mode - a new pairwise independent key is generated in each iteration by adding a Initial Value modulo a prime $p$ in GF(256)

iteration of the BCAC coder using an initial key value. The same values can be reconstructed in the decoder side with prior knowledge of these initial values. However, the generated key values are pairwise independent from each other. This method uses Galois field mathematics and we take $3N \leq 256$ or $N = 85$ for BCAC to simplify the operation. The generated keys are shown to be pairwise-independent by Jutla et al. [14]. There is no loss in compression efficiency or throughput of the encoder. See Algorithm 3 and Figure 3(d) for details. This mode has the restriction that $N$ is should be keep to a value such that $3N \leq 256$ or an exponent of 2 (for efficient finite field implementation).

### *D. Resistance to Known Attacks*

Assessing security for any encryption system is a challenging task because showing robustness against known attacks does not preclude the existence of unknown attacks against which the system may not be robust. This applies to mature encryption standards such as AES [15] and DES [16] also. We therefore adopt a similar approach that considers known attacks and ensures that they cannot be used successfully.

One great security advantage of our presented scheme is that the output from the engine is in the form of variable sized words and the individual bit output corresponding to inserted symbols cannot be determined.

The decoding algorithm for CAC involves iteration on chaotic maps. Kocarev [17] discusses how various properties of chaotic maps have direct correlation to cryptographic algorithms. For example - decoding CAC with any slightly wrong value (making a wrong guess) will make the output appear random even if correct knowledge of maps is given. This is analogous to diffusion property of cryptographic ciphers. Similarly, the iterations on chaotic maps (1 iteration per encoded bit) is similar to rounds in encryption algorithms. Thus, the chaotic decoder will behave like a random number generator and without exact knowledge of key (coding parameters) and initial seed (coded message), the output of decoder will be completely un-correlated with the encoded message. This property of chaotic maps implies that unlike BCAC, two closely related plaintext values may be mapped to completely different (random) output values even with same key. And the same message, will be mapped to completely different output value with two closely related key values.

This makes it difficult to launch **known-plaintext attacks** on our system. It is however possible to mount **chosen-plaintext attacks** on the system because an attacker can modulate the plaintext inputs to iteratively guess the key stream beginning from first bit of plaintext (last bit of BCAC encoder). Such attack has been mounted against KSAC [6].

The proposed SEs can alleviate the attacks at little computational overhead. For BCAC+PM, the first few bits of plaintext are modulated, these bits and the key bits are unknown to the encoder. Therefore, it will be impossible for an attacker to observe and infer any correlation from chosen-plaintexts. In BCAC+Mix operation, at every iteration we XOR the key with the output of encoder and update the key. This randomizes the output (like a one-time pad) for an adversary to draw any inference. The BCAC+PWIK mode allows us to resist chosen and known plaintext attacks because the keys used in different iterations are pairwise independent, hence, an attacker cannot find any correlation between subsequent output bits corresponding to same plaintext value. However, it comes with an extra implementation cost of PWI Key generation module. Either of the two proposed modes (Mix and PWIK) have no effect on compression efficiency, which is a significant advantage against some proposed techniques [4], [18], [10]. A drawback, of PWIK mode is that it involves GF mathematics: the length of input bits should suit the GF operations. For example, with GF($2^{256}$) implementation, the length of plaintext will be 85 bits.

BCAC, like arithmetic coding, is more sensitive to errors in the decoded bitstream for errors in the beginning of the stream and not to those which are towards the end. However, BCAC+Mix mode has bit rotate and XOR operations which mask this property from the adversary.

### *E. Comparison with BAC+AES*

BAC followed by encryption with AES is the naive candidate which should provide best security. AES is extremely fast when it is fully pipelined in hardware [19]. However, the sequential nature of BAC coder becomes the bottleneck in a combined BAC+AES system.

The arithmetic operations required for one bit encoding or decoding using BAC is 2 adders and 1 multipliers (discussed in Section II-B). AES-128 requires 40 sequential transformation steps composed of simple and basic operations such as table lookups, shifts, and XORs. It needs approximately 336 bytes of memory and approximately 608 XOR operations or roughly 3 bytes memory and 5 XOR operations per bit of encoding.

BCAC coder requires 2 adders and 2 multipliers for encoding and only 1 adder and 1 multiplier for decoding. Thus, the hardware requirements of BCAC coder are much less than BAC and AES combined. The BCAC decoder is particularly simpler than CAC decoder (without AES), which is desired for most common video applications which involve real-time decoding in mobile and embedded devices.

### *F. Key and Plaintext Sensitivity*

Confusion and diffusion are two important properties desired for operation of a secure cipher. Confusion refers to making the relationship between the key and the ciphertext as complex and involved as possible. This makes it very hard to find the key even if one has a large number of plaintext ciphertext pairs produced with the same key. In particular,

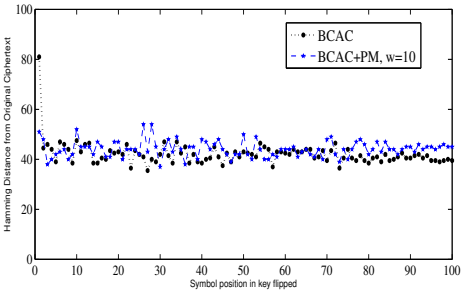


Figure 4. Plot showing the number of bit-flips in output text with change in one corresponding symbol in the key

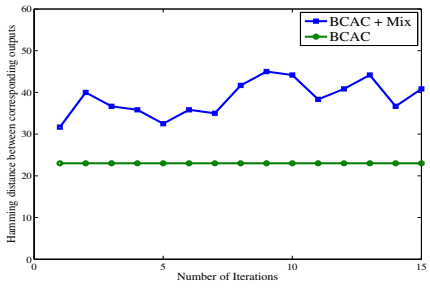


Figure 5. Plot of hamming distance between correspondingly coded values when a single bit of key ($N = 50$) was changed.

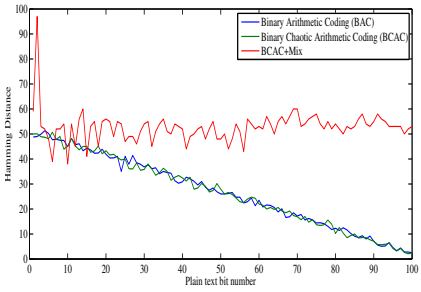


Figure 6. Plot showing the number of bitflips in output text vs change in a single bit position in the plaintext

changing one bit of the key should change the ciphertext completely.

We conducted experiments for different values of $N$ by changing one bit in either of the $N$ symbols comprising the key. In the ideal case, for a single bit flip, the ciphertext output must be changed from original value in $\frac{N}{2}$ positions, as was the case with BCAC. Figure 4 shows the plot for $N = 100$ and $p = 0.7$ with mean value of 1000 simulations. BCAC, BCAC+PWIK and BCAC+Mix have same encoder output for the first iteration, so we have used a single line to represent this. It can be seen that all the schemes (including BCAC+PM) give a hamming distance of new ciphertext to a value around 50. Over different iterations, BCAC+Mix will eventually lead to different key being used in different iterations, leading to a variable hamming distance. This increases the confusion property of the scheme and it is shown in Figure 5.

Diffusion means that the output bits should depend on the input bits in a very complex way. This is ensured by the arithmetic coding (or CAC) scheme itself because the relationship between input and output bits is non-linear. The input bits iteratively decide the interval of final output, which is then used to obtain the shortest length code from that interval. However, it has been observed in case of BAC that the last few bits have less impact on the first bits of the ciphertext. We conducted an experiment where plaintexts were varied one bit at a time and hamming distance of new ciphertext over last ciphertext was reported. The mean value over 1000 such simulations is reported in Figure 6. BAC and BCAC show similar trend vs. change in plaintext bits. BCAC+PWIK, BCAC+PM have similar performance trend as BCAC.

BCAC+Mix mode result in mixing of encoder output with the key value and this mixing leads to a excellent value of hamming distance, as plotted in Figure 6.

### G. Selective Encryption using BCAC

In this section, we present results for selective image encryption using BCAC. The sample images were taken from the USC SIPI database and each has resolution 512x512 pixels. Two cases are considered: (a) Encryption of LL coefficients and (b) all coefficients of 6th level DWT decomposition. This corresponds to encryption of $0.1\%$ and $0.4\%$ coefficients. SSIM (Structural Similiarity Index) and PSNR (Peak Signal to Noise Ratio) were considered to measure the image reconstruction with corrupted keys. The results are presented in Table II. It can be observed that SSIM - Structural Similarity is close to 0 while PSNR is negative, which indicate strong de-correlation between wrongly reconstructred image with original image.

Table II
RESULTS FOR IMAGE RECONSTRUCTION QUALITY WITH WRONG DECODING KEYS. SELECTIVE ENCRYPTION OF DWT COEFFICIENTS WAS DONE USING BCAC.

| Image | $0.1\%$ encryption | | $0.4\%$ encryption | |
|---|---|---|---|---|
| | SSIM | PSNR | SSIM | PSNR |
| Tank | 0.057 | $-6.32$ | 0.00 | $-11.55$ |
| Couple | $-0.23$ | $-6.56$ | $-0.03$ | $-8.5$ |
| Girl | $-0.05$ | $-4$ | 0.005 | $-7$ |
| Grass | $-0.08$ | $-6.57$ | 0.002 | $-11.43$ |
| Peppers | $-0.06$ | $-7.936$ | 0.012 | $-10.51$ |
| San Diego | 0.111 | $-6.2$ | 0.06 | $-10.22$ |
| House | 0.16 | $-4.17$ | .035 | $-4.21$ |

## IV. COMPRESSION

BCAC gives the same compression efficiency as the BAC coder. We performed some experiments to verify these facts. We ran an implementation of BCAC over Matlab R2010b and used variable precision arithmetic (vpa) tools in the Symbolic Mathematics Toolbox to run simulations for large values of $N$ (such as $N = 100, 1000$).

The simulation results show a slightly better performance for CAC over normal arithmetic coder (AC) especially for small values of $N$. However, as mentioned above there is no objective reason for such occurrence. The results are presented in Table III (The reported value is the average length of output bitstream and the standard deviation). 1000

Table III
COMPRESSION PERFORMANCE OF BAC AND BCAC FOR VARIOUS LENGTH STRINGS. THE AVERAGE LENGTH OF CODEWORD IS PRESENTED FOR VARIOUS $p$ VALUES AND LENGTHS OF INPUT STRING.

| $p$ | $N = 10$ | | $N = 100$ | | $N = 1000$ | |
|---|---|---|---|---|---|---|
| | BAC | BCAC | BAC | BCAC | BAC | BCAC |
| 0.5 | 0.025 | 8.733 | 120 | 108.16 | 999.1 | 999.84 |
| 0.55 | 9.025 | 8.983 | 98.17 | 98 | 992.30 | 992.34 |
| 0.6 | 8.7899 | 8.882 | 95.95 | 95.84 | 970.98 | 971.14 |
| 0.65 | 8.442 | 8.316 | 91.90 | 91.23 | 934.30 | 934.04 |
| 0.7 | 7.918 | 7.936 | 86.96 | 86.37 | 881.07 | 881.84 |
| 0.75 | 7.47 | 7.54 | 80.31 | 80.37 | 811.27 | 811.84 |
| 0.8 | 6.701 | 6.333 | 71.11 | 71.03 | 721.07 | 720.84 |
| 0.85 | 5.551 | 5.342 | 60.28 | 59.1 | 609.30 | 609.84 |
| 0.9 | 4.122 | 4.055 | 45.66 | 46.39 | 469.06 | 468.84 |
| 0.95 | 2.698 | 2.773 | 27.46 | 28.8 | 287.00 | 286.34 |

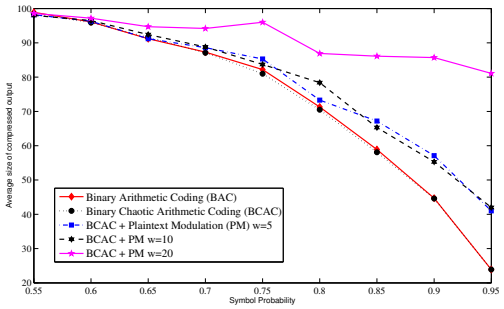


Figure 7. Compression performance of proposed schemes

simulations each were run in Matlab to obtain the mean value of output bitstream lengths.

Figure 7 gives the relative compression performance of CAC, BCAC and various SEs (for $N = 100$). BCAC+PM has a slight compression overhead for $w = 5$ or 10, but it increases drastically for $w = 20$, making $w = 20$ unsuitable for practical applications. PWIK and MIX modes (not shown in this figure) have similiar compression as BCAC.

## V. CONCLUSION

In this paper we presented a joint compression and encryption scheme for video/ images using chaotic maps. We presented some SEs to alleviate the weaknesses of presented scheme against known cryptanalysis.

The presented scheme incurs no loss to compression performance, has a simpler decoder while at the same time it encrypts data. It was shown to achieve higher throughput than the naive encryption algorithms.

## VI. ACKNOWLEDGEMENT

This work is partially supported by the National Science Foundation under Grant #1019343 to the Computing Research Association for the CIFellows Project.

## REFERENCES

[1] G. Langdon and J. Rissanen, "Compression of black-white images with arithmetic coding," *IEEE Trans. Communications*, vol. 29, no. 6, pp. 858–867, Jun 1981.

[2] Y. Mao and M. Wu, "A joint signal processing and cryptographic approach to multimedia encryption," *IEEE Trans. Image Processing*, vol. 15, no. 7, pp. 2061–2075, July 2006.

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

[4] H. Kim, J. Wen, and J. Villasenor, "Secure arithmetic coding," *IEEE Trans. Signal Processing*, vol. 55, no. 5, pp. 2263–2272, May 2007.

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

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

[7] J. Zhou, O. C. Au, X. Fan, and P. H. W. Wong, "Joint security and performance enhancement for secure arithmetic coding," in *ICIP*, 2008, pp. 3120–3123.

[8] J. Zhou, O. C. Au, P. H. Wong, and X. Fan, "Cryptanalysis of secure arithmetic coding," in *ICASSP*, 2008, pp. 1769–1772.

[9] H.-M. Sun, K.-H. Wang, and W.-C. Ting, "On the security of the secure arithmetic code," *IEEE Trans. Information Forensics and Security*, vol. 4, no. 4, pp. 781–789, 2009.

[10] N. Nagaraj, P. Vaidya, and K. Bhat, "Arithmetic coding as a non-linear dynamical system," *Communications in Nonlinear Science and Numerical Simulation*, vol. 14, no. 4, pp. 1013–1020, 2009.

[11] A. Pande, J. Zambreno, and P. Mohapatra, "Joint video compression and encryption using arithmetic coding and chaos," in *IEEE International Conference on Internet Multimedia Systems Architecture and Application*, 2010.

[12] H. Yu, "Scalable encryption for multimedia content access control," in *IEEE Intl. Conf. Multimedia and Expo, ICME 2003*, vol. 1, july 2003, pp. I – 633–6 vol.1.

[13] P. Moo and X. Wu, "Resynchronization properties of arithmetic coding," in *IEEE Intl. Conf. Image Processing, ICIP1999*, 1999.

[14] C. S. Jutla, "Encryption modes with almost free message integrity," in *EUROCRYPT*, 2001, pp. 529–544.

[15] FIPS 197, "Announcing the Advanced Encryption Standard," Nov 2001.

[16] FIPS 46-2, "Announcing the standard for Data Encryption Standard," Dec 1993.

[17] L. Kocarev, "Chaos-based cryptography: a brief overview," *Circuits and Systems Magazine, IEEE*, vol. 1, no. 3, pp. 6–21, 2002.

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

[19] J. Zambreno, D. Nguyen, and A. N. Choudhary, "Exploring area/delay tradeoffs in an AES FPGA implementation," in *Proc. IEEE Intl. Conf. Field Programmable Logic and Applications, FPL 2004*, 2004, pp. 575–585.