# Partial Encryption of Video Bit-Streams using Coupled Chaotic Maps

***Abstract*— Due to pervasive communication infrastructures a plethora of enabling technologies is being developed over mobile and wired networks. Among these, video streaming services over IP are the most challenging in terms of quality, real-time requirements, and security. In this paper we propose a novel scheme to efficiently secure variable length coded (VLC) multimedia bit streams such as H.264. It is based on code word error diffusion and variable size segment shuffling. The codeword diffusion and the shuffling mechanisms are based on random operations using a computationally efficient secure chaotic maps random number generator. The proposed scheme is ubiquitous to the end users and can be deployed at any node in the network. It also works on the compressed bit stream without requiring any decoding. The proposed scheme is 200% faster, and 150% more power efficient when compared with AES-based software full encryption schemes. Regarding security, the scheme is robust to well-known attacks in the literature such as brute force and known/chosen plaintext attacks.**

***Keywords-component; RTP, Video encoding, VLC, Huffman code, chaotic maps***

## I. INTRODUCTION

Due to recent developments in the field of multimedia communications, applications such as Voice over IP (VoIP), video conferencing, e-learning and digital TV/HDTV are now part of everyday life. We are immersed in a worldwide information network where people do business online, and have access to news, bank accounts, etc., from their offices or homes. These digital commodities have some inherent risks; communication networks (wired/wireless) are vulnerable to attacks violating the user's right of privacy. It is imperative to design fast and secure systems. However, in the case of multimedia data, security demands addressing new challenges, primarily due to sheer volume of data involved, temporal dependent nature of the information, and time processing restrictions for real-time multimedia communications. The problem of security for mobile communications is further exacerbated by the limited processing power and battery life available in diverse devices, particularly handheld or mobile, for which provisioning of security may be infeasible when the complexity of related decoding operations is over the processing limit of such devices. Therefore security solutions have to be power efficient to allow longer and more number of sessions between mobile users within a single battery charging cycle. Any solution involving full encryption of the information limits itself in terms of scalability. Furthermore, it is desirable that the solution should work in the compressed domain without requiring decoding of the bit stream. In this work, we focus on the encryption of partial contents rather than encrypting the complete multimedia coded bit stream. Our solution is a novel scalable technique to secure variable length coded (VLC) bit streams aimed at minimizing the encryption complexity while preserving security. It can be implemented at any stage of the communication pipeline, that is, after the encoding process on the sending device (handheld or mobile device) or at a dedicated server handling multiple multimedia streams.

Numerous data encryption schemes have been proposed for multimedia streaming applications that lower the complexity by selectively encrypting the data. Khanvilkar et al. [10] proposed a selective encryption approach for mp3 bitstreams that partially encrypted selected fields in the mp3 header. Meyer and Gadgast [15] proposed a selective encryption scheme for MPEG-1 bit streams. The principle data to be secured included: all the headers, I frames, and I blocks. They proposed a number of combinations of the above scheme to attain different levels of security. Spanos and Maples [19] proposed to encrypt the I frames and the ISO start and end code of the MPEG stream. Tang [22] proposed an approach to use random permutation list instead of the zigzag order for mapping an 8x8 DCT block to a 1x64 block. Without the actual permutation list it would be difficult to perform the inverse DCT transform on this data. This approach yielded non-optimal compression. Liu and Eskiciogla [11] gave a comparison between the traditional and selective encryption approach, and showed that selective encryption based techniques suffer in one or more of the following points:

1) Insufficient security
2) Decrease in the compression performance of entropy coding
3) Insignificant computational reduction with respect to total encryption
4) Lack of bitstream compliance
5) Increase in key size
6) Require compression decoding

Wu and Kuo [25] proposed a scheme that performs both compression and encryption by using multiple Huffman tables in the entropy encoder. The secret key used for encryption and decryption consists of G distinct Huffman coding tables. Huffman tables are then selected randomly from some public pool of Huffman tables. Wen et al. [24] proposed a binary arithmetic coding with key interval splitting. The proposed scheme is designed to achieve both compression and confidentiality by splitting the intervals according to a secret key. Jakimoski and Subbalakshmi [9] gave a cryptanalysis of different multimedia encryption schemes. He showed that [24] and [25] are vulnerable to low complexity known- and/or

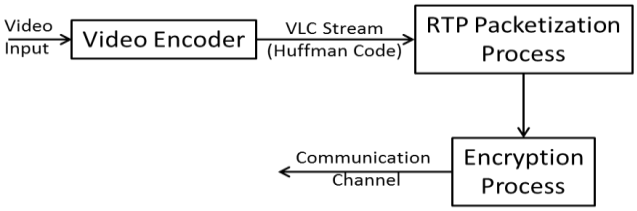


Figure 1. General steps of the video data transformation process.

chosen-plaintext attacks. Different than the previous codec dependent security schemes, our aim is to secure VLC coded bit streams including mp3, H.263, H.264 codec through random bit changes and by generating decoding errors that propagate as long as the resulting codewords are valid. The longer the propagation error, the more secure is the system. However several works have been proposed to minimize the effect of error propagation on subsequent codewords. In [2, 3, 7, 14, 17, 23], solutions have been investigated to overcome such errors by exploiting correlation among the codewords. A straight forward solution is to drop the corrupted sequence once it is detected, and resynchronize from the new synch position within the bit stream. The research works involving self-synchronizing codes that quickly help reestablish synchronization and thus reduce the run length of error propagation can be found in [2, 3, 7, 14, 17, 23]. A more complex solution is to approximate the corrupted coefficient value from previous or/and advance values. Te-Chung and Kumar [23] have studied the dependency in inter-subband coefficient values, and showed that there exists certain correlation between coefficient of parent and child subbands of a picture. Based on the subbands coefficient relation, they have proposed a scheme to recover the lost values. The scheme divides the corrupted subband into three regions, the correctly coded region, the error propagation region and the shifted region. The scheme transmits two parameters within the compressed stream to be used at the decoder stage to reconstruct the corrupted values. These types of schemes work under the assumption of isolated bite errors for which the upcoming bit sequence can be resynchronized. For additional details on security techniques related to VLC bit streams we refer to [13-14].

In this paper we focus our work on variable length coded (VLC) video streams. The main idea of our encryption scheme is to make the decoding of the VLC codewords in the bit stream computational infeasible in the absence of a secret key. Assuming video stream consists of packets, each packet is divided into random size segments. Within each segment, bits are randomly flipped such that the correlation present among codewords is diffused. Then all the segments are randomly shuffled. The randomness of bit flipping and shuffling of segments is based on a secure random number generator. For consecutive packets the shuffling and flipping patterns are completely different, and are chosen based on unrelated random numbers that are computationally infeasible to guess from the secret key. We realize such a robust and secure random number generator using coupled chaotic maps. Our proposed scheme is 200% faster, and 150% more power efficient when compared with AES based full encryption schemes, and secure against common attacks such as brute force and known/chosen plaintext attacks.

The rest of this paper is organized as follows: In the next section we describe major components of the proposed encryption scheme. Section 3 analyzes salient characteristics of the proposed scheme. In section 4, security analysis and comparisons with existing schemes are provided. Performance evaluation setup and experimental results are discussed in sections 5 and 6. Conclusions of the work are presented in section 7.

## II. PROPOSED SCHEME

The encryption process begins right after the RTP packetizing process (Figure 1), in which video payload (VLC bitstream) varies from 300 bytes to 1400 bytes depending on the transmitted media properties. Our proposed scheme encrypts VLC coded bitstream by performing the following dynamic operations on every RTP packet: random bit(s) flipping and segment shuffling. The position of the bits to be flipped, locations of the segments to be shuffled, and segment size depend on a secure pseudo random number generator. A block diagram of the proposed scheme is given in Figure 2. It is composed of three main blocks: A) Secure Random Number Generator Based on Coupled Chaotic Maps, B) Bit Flipping, and C) Segment Shuffling.

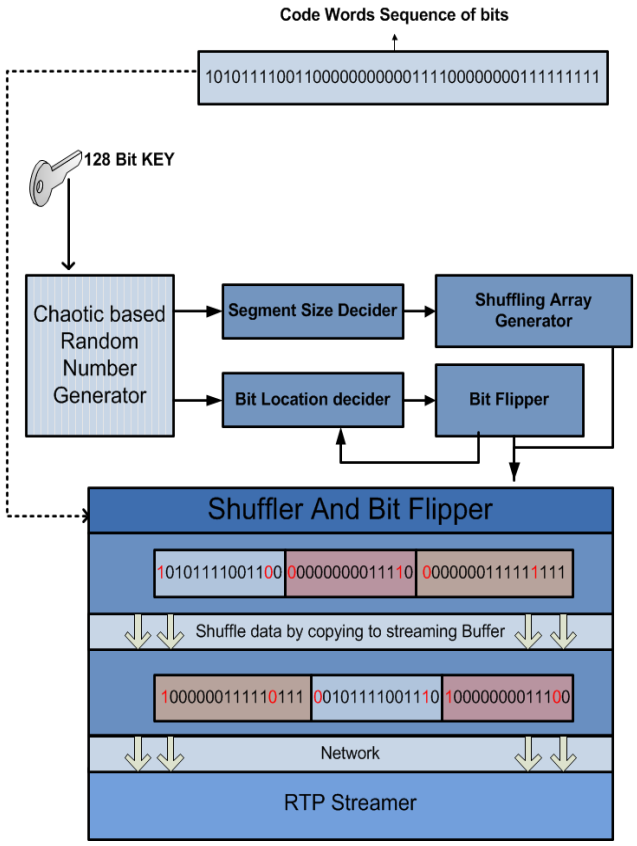


Figure 2. Illustration of the proposed encryption scheme

### *2.1 Secure Random number generator based on chaotic maps*

The security of the proposed scheme to different attacks depends mainly on the robustness of the secure Pseudo-Random Number generator (PRNG). The scheme can work with any secure PRNG as long as the seed cannot be determined from a partially broken sequence generated. Current PRNG are not strong candidates to be included in our scheme because of their dependency on a fixed-length seed, as well as the lack of flexibility to dynamically control the security of the system. We develop a novel PRNG based on a network of *N* chaotic maps, that is *N* discrete chaotic maps dynamically interacting as one system but maintaining their own identity (the use of only one chaotic map does not provide enough security to the system). Discrete Chaotic Systems (DCS) have many of the good properties required in cryptography; the most prominent are sensitivity to parameters, sensitivity to initial conditions and unpredictable trajectories [15]. The first two properties are related to diffusion, and the last one to confusion in the cryptographic nomenclature. Confusion is intended to make the relationship between cipher text and plaintext statistically independent, whereas diffusion is intended to spread out the influence of a single plaintext digit over many cipher text digits to hide the statistical structure of the plaintext. These properties have been the basis to develop secure analog and digital communication systems.

Current research in chaotic systems is focused on two main issues: Perturbation-based schemes and Network-based chaotic maps. Perturbation–based schemes transform stable chaotic cycles into non-stable ones by perturbing its trajectory as performed in [15]. A network of chaotic maps or Coupled Map Lattices (CML) on the other hand considers an array of chaotic maps governed by a coupling transformation over some defined neighborhood in the array [5]. In this work, we use CML to develop a PRNG that is robust to cipher text, known/chosen plaintext and differential attacks.

Our proposed PRNG is based on a network of *N* chaotic maps ($1 \le i \le N$), represented by:

$$X_{i,j} = (1-\varepsilon) f\left(X_{i,j-1}\right) + \varepsilon H\left(X_{i,j-1}, \dots, X_{N,j-1}\right)$$
$$H\left(X_{i,j-1}, \dots, X_{N,j-1}\right) = \sum_{i=1}^{N} w_i X_{i,j-1} \qquad (1)$$

The states $j$ in the chaotic network represent the weighted interaction between each individual map $f(X_{i,j-1})$ (local term) and the coupling transformation $H$ (linear/nonlinear interaction term) with weights $w_i$, such that $\sum_{i=1}^{N} w_i$, and $N \ge 8$. When the weight $\varepsilon$ is weak (small magnitude), the system can be regarded as a local map perturbed by contributions from other sites, thus maintaining its main individual properties. On the other hand, when $\varepsilon$ is large, the system reaches an asymptotic collective (undesired) behavior characterized by intermittent periodic chaotic cycles (this is the case we want to avoid).

For its mathematical simplicity our selection for $f(X_{i,j})$ is the well-known logistic map represented by:

$$X_{i,j} = f(X_{i,j-1}) = \lambda X_{j-1}(1 - X_{j-1}),\ \lambda \in [1,4),\ X \in (0,1) \qquad (2)$$

Where $\lambda$ represents the chaotic parameter and $X$ the state variable. As $\lambda$ increases from 1 to 4, the map experiences a period doubling to chaos. In particular for $\lambda \ge 3.5699$ (known as accumulation point) it presents a chaotic behavior, however there are many undesired periodic windows with short periods that appear abruptly. A very well-known and prominent period-3 window appears at $\lambda = 1 + \sqrt{8} = 3.8284$. Fixed points $(f(X) = X)$ are also present at X = 0 and X= (λ -1)/ λ, which define a regular pattern in the logistic map. In order to keep the good chaotic dynamic of the logistic map, we avoid bad initial values of X and λ, and endorse the use of at least 8 chaotic maps to increase the cycle length period or the use of cycle-length tracking schemes as the one proposed in [6]. It is important to point out that any chaotic map in the literature (Renyi Map, Piece-wise Linear map, etc.) can be used in Eq.1. The security relies on the scheme itself, rather than the chaotic map used in the scheme.

As mentioned before, the main reason for developing our own PRNG scheme is to manage the security of the system by creating a (near) cycle free chaotic signal capable of handling long-term multimedia communications such as VOIP and video streaming (which can last from minutes to hours). The security is controlled by changing the number of chaotic maps and periodically perturbing the system state (variables, parameters, coupling function, weight $\varepsilon$, etc.). Additionally, we include previous input data P (called plaintext) as part of the coupling function $H$ to allow diffusion of the information onto the entire ciphertext output as follows:

$$H_j\left(X_{1,j}, \dots, X_{N,j}, P\right) = \sum_{l=1}^{N} w_l X_{l,j} + w_{N+1} \mathrm{P}'_j \qquad (3)$$

$$\mathrm{P}'_j = \frac{\left(\sum_{l=1}^{M} P_{j-1,l}^{32}\right)\ mod\ \mathrm{H}'_{j-1}}{\mathrm{H}'_{j-1}}$$

Where $\sum_{l=1}^{32} P_{j-1,l}^{32}$ represents the sum of all 32-bit plaintext variables in the previous RTP packet and $H'_{j-1}$ is the integer representation of the previous coupling function. A bit change in P affects the outcome of the bit-flipping and shuffling operations of future iterations proportionally to the magnitude of $W_{N+1}$ and ε. Even though the computational complexity of the scheme is slightly increased, a single plaintext change produces a totally different ciphertext, therefore increasing its robustness to statistical attacks.

To further protect the system against security attacks, the following actions are considered:

1) Every iteration in Eq.3 produces N 32-bit chaotic trajectories (or pseudo-random numbers) coming from randomly selected chaotic maps. That is, previously evaluated chaotic map trajectory in Eq.3 determines the next map to be iterated (*next_map = previous_chaotic_trajectory mod N*). In the case of an attack, the random selection of maps increases the complexity considerably.

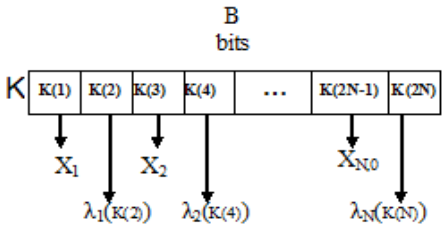


Figure 3. System-key (K) partition (n/2) for the creation of maps variables (Xi,0) and corresponding parameters($\lambda_i$).

2) The actual random number to be used in the encryption process is produced by operating three different maps according to the following formula:

$$Out_i = \left[\left(X'_{i,j} + X'_{i+1,j}\right) mod\ 2^{PR}\right] \oplus X'_{i+2,j} \qquad (4)$$

Where PR is the computer precision (32 or 64 bits), $\oplus$ is the XOR operator, and $X'_{i,j}$ is the integer part of $X_{i,j}$ ( $X'_{i,j} = \mathrm{INT}[X_{i,j} * 2^{32}]$) .

3) Only $1 \le U \le 27$ bits of $Out_i$ per map are taken into account in the encryption process; the remaining L=U-27 bits are used for future encryption in a randomly selected iteration. These actions prevent the attacker from having complete knowledge of the system even when the security of state variables $X_i$*'s* has been compromised (see section V). Equations 1-4 form the Chaotic-based Random Number Generator block in Figure 2, and are encapsulated in function *rnd (U)*, which returns a U-bit random number $Out_i$ from the *N* 32-bit random numbers generated.

A.1 *Chaotic System Generation*

The proposed scheme is symmetric, therefore the initial system-key (*K*) of size *B* bits, for $B \ge 128$ is shared between cipher and decipher. Any suitable key establishment/distribution protocol (public-key or authenticated protocol) in the literature can be used for the key exchange, the only restriction is that every session requires a new and independent system-key.

The system-key *K* is used to initialize the *N-array* of chaotic maps (eq. 5) as follows (see Figure 3):

$$X_{i,0} = K_{n/2}(2i-1)/2^{n/2},$$
$$\lambda_i = 3.68 + \frac{[K_{n/2}(2i)/2^{n/2} + K_{n/2}(2i)/10^{h_{n/2}} + (a \oplus b)/2^{n/4}]}{10} \cdot \frac{[0.3187]}{MAX},$$
$$i = 1,2,\ldots,N \qquad (5)$$

where $n=B/N$ is the assigned number of bits per map (taken from *K*) for the initialization of $\lambda_i$ and $X_{i,0}$ (*n/2* bits per variable and parameter) with $N << B$ and ($B$ *module* $N$) = 0, $K_{n/2}$(m) represents the system-key as an array of *n/2* bits per element, $h_{n/2}$ is the number of digits in the largest decimal number represented by *n/2* bits, $a \oplus b$ term is the XOR between the half-most and half-least significant bits of $K_{n/2}(2i)$ respectively, yielding an *n/4* bits outcome, and MAX is the maximum value of $[K_{n/2}(2i)/2^{n/2} + K_{n/2}(2i)/10^{h_{n/2}} + (a \oplus b)/2^{n/4}]/10$.

The chaotic variables and parameters are forced to fall in the range $X_{i,0} \in (0,1)$ except $\{\lambda/(\lambda-1), 0.5\}$, which represent bad initial points) and $3.68 \le \lambda_i \le 3.998$ for $\lambda_i \ne \lambda_j$ and $i \ne j$ respectively. The remaining variables in eq. 5 $\varepsilon$ and $w_i$, $1 \le i \le N$ are initialized by iterating N+1 times a predefined map $X_P$ in the chaotic system. The first *N* chaotic values are used to compute $w_i = X_{P,i} \Big/ \sum_{j=1}^{N} X_{P,j}$ , and the last value to compute $\varepsilon = \Delta\varepsilon \cdot X_{p,N+1} + \varepsilon_{min}$, which represents a linear transformation from the chaotic space $X \in (0,1)$ onto the $\varepsilon \in (\varepsilon_{min}, \varepsilon_{max})$ with $\Delta\varepsilon = (\varepsilon_{max} - \varepsilon_{min})$, $\varepsilon_{min}$ and $\varepsilon_{max}$ the allowable minimum and maximum value of $\varepsilon$ respectively.

After the chaotic system has been created, we perform an additional operation to increase its sensitivity to bit changes *K* (if *K* is changed by one bit, the new chaotic system will differ significantly from the original one). The original initial chaotic variable $X_{i,0}$ is iterated random number of times, say *R*, over the newly created coupled chaotic system, and the corresponding output becomes the initial state for each map in the encryption process. The same process can be performed for the corresponding map parameters ($\lambda_i$), if needed, using the new variables.

The system-key *K* is used to define the number of chaotic maps to be used in Eq. 5 as follows:

$$N = [KS\ mod\ (MAX_B + 1)],$$
$$KS = \sum_{i=1}^{D/8} KEY_8(i) \qquad (5)$$

where $MIN_B \le N \le MAX_B$ following Table II.

**TABLE II.** Minimum and maximum number of chaotic maps involved in the encryption process as a function of the system-key length.

| KEY LENGTH (BITS) | MINIMUM NUMBER OF MAPS ($MIN_B$) | MAXIMUM NUMBER OF MAPS $MAX_B$ |
|---|---|---|
| 128 | 8 | 16 |
| 256 | 8 | 25 |
| 512 | 16 | 32 |

### *2.2 Bit flipping*

Once the encryption system receives an RTP packet, the main step in our scheme is to diffuse and destroy the meaning of compressed codeword sequence and make it impractical to predict the original codeword sequence. To achieve this, we propose the flipping of at least one bit every $BF = f \cdot Av \cdot MEPL$ bits, where *Av* is the average size of Huffman codes, MEPL is the calculated Mean Average Propagation Length (MEPL) in codeword units (described in section III), $Av \cdot MEPL$ is the average error propagation in bits, and $f$ is a tunable security factor with values $1/(MEPL) \leq f \leq \frac{3}{4}$. For this range of *f*, $Av \leq BF \leq (A_v \cdot MEPL)$ that is, at least one bit is changed per average Huffman codeword size (*Av*) up to $\frac{3}{4}(A_v \cdot MEPL)$ bits. This is what makes our *system scalable secured*, as *f* gets smaller more bits are flipped per BF-bits units increasing the system security. Depending on the specific needs of the user, $f$ provides a tradeoff between security and performance.

The actual location *Bi* of the bit to be flipped in the stream is calculated as follows:

$$Bi = [rnd(\lceil(Log_2(f \cdot Av \cdot MEPL)\rceil) \bmod BF] + 1 \quad (6)$$

where *rnd(#bits)* is our own designed PRNG function described in the previous section. The argument $\lceil Log_2(f \cdot av \cdot MEPL)\rceil$ represents the bit-length of the requested random number with bounds $1 \leq B_i \leq BF$.

The algorithm for random bit flipping in the payload P of size Ps bytes is outlined in the following:

1. *BF = f·Av·MEPL*
2. For i =0 to (*Ps*8) / BF* steps do
3. $Bi = (rnd(\lceil Log_2(BF)\rceil) \bmod BF) + 1$
4. loc = i * (*BF*) // **next partition of length BF**
5. Flip ( P[Bi + loc] )
6. Repeat

### *2.3 Segment shuffling*

After the bit flipping process, we permute the RTP-packet payload by performing an L-way shuffling. Here the payload is divided into L segments which are shuffled using the algorithm described below and illustrated in Figure 4. The complexity of the shuffling (L!) is expressed in terms of $2^S$, where $S$ is a user controlled variable that specifies the minimum brute force complexity required to de-shuffle the L segments. L is obtained as follows:

$$L = (rnd(\lceil log_2(L_{max})\rceil) mod (L_{max} - L_{min})) + L_{min} \quad (7)$$

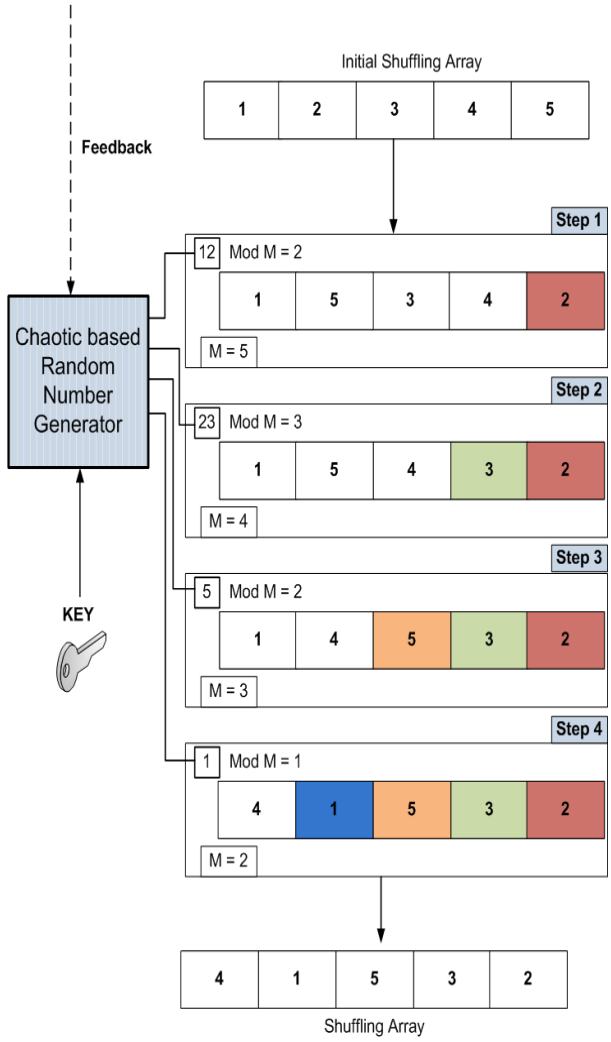


Figure 4. Illustration of Building the shuffling array for an RTP packet with L=5 segments.

where *Lmin* and *Lmax* are the smallest integers satisfying $L_{min}! \geq 2^S$ (for a given user input value S) and $L_{max}! \geq 2^S$ respectively. For each iteration (or RTP Packet) the number of segments is a random number between $20 \leq L \leq 60$ which corresponds to $60 \leq S \leq 272$ (which increases the security of the system at user´s needs).

Once L has been computed, the segment size Sg is calculated using Eq.8:

$$Sg = \frac{Ps}{L} \quad (8)$$

where *Ps* is the RTP payload size in bytes. For a 300-byte RTP packet the segment size randomly falls between $5 \leq Sg \leq 15$, which corresponds to $20 \leq L \leq 60$ segments.

The proposed shuffling mechanism starts by creating the shuffling array A of size L that will be used to shuffle the segments in O(L!) steps as illustrated in Figure 4. Each entry in the initial shuffling array is the original block index. The

scheme will iterate L-1 times, starting by initial value of M = L. The following algorithm generates the shuffling array:

1. Initialize A // **shuffling array with L elements**
2. M = L
3. For L- 1 steps do
4. Generate random number R
5. T = R mod M
6. Swap (A[T ] , A[M])
7. M = M – 1
8. Repeat

The resulted entries in array A are used to determine the new destination of each segment; the array elements are moved from current location T to final location M in the same array.

## III. System Analysis

The key feature in our encryption is to make the reading and decoding of Huffman codes impossible without knowing the private key. The key characteristic exploited here is the prefix condition of the Huffman codes, which states that no code words can be a prefix of another code word [8].

The prefix code condition is used to guarantee the unique parsing path for each codeword, as shown in Figure 5. This condition provides hidden (implicit) borders that separate codeword's from each other. As we see in Figure 5, to satisfy the prefix code condition, each codeword ends at a leaf node in the tree. If any bit in a code word is flipped (shown with red arrow in Figure 5, then a transition will occur at that point which results one of the following scenarios:

1) Different codeword same length size.
2) Different codeword shorter length size.
3) Different codeword longer length size.

The first scenario will only change the codeword which includes the flipped bit. This is unlikely to happen frequently in VLC; as the length of codewords is variable. However, the effect of the corrupted codeword in this scenario on the subsequent codewords depends on the information carried out by the corrupted codeword. For example, if the codeword holds the DC value of the macroblock, the macroblock will not be decoded correctly as well as all subsequent blocks. But if the codeword holds the AC value of the macroblock, only that macroblock will not be decoded correctly. There are many candidates of what value that codeword can represent, such as DC, AC, MV, header, etc.

The second and third scenarios will break the hidden borders between codewords, thus making the reading process misaligned and incorrect as shown in Figure 6. The error will propagate at least to the next codeword, where one of the previous scenarios can happen again. The chain of second and third scenario occurrences is called error propagation. The run length of the error propagation is at least one codeword on average. Furthermore, the mean error propagation length MEPL (introduced in Section 2.C) for a particular bit error that occurred in a particular codeword is defined as the number of codewords in the sentence from the one where the

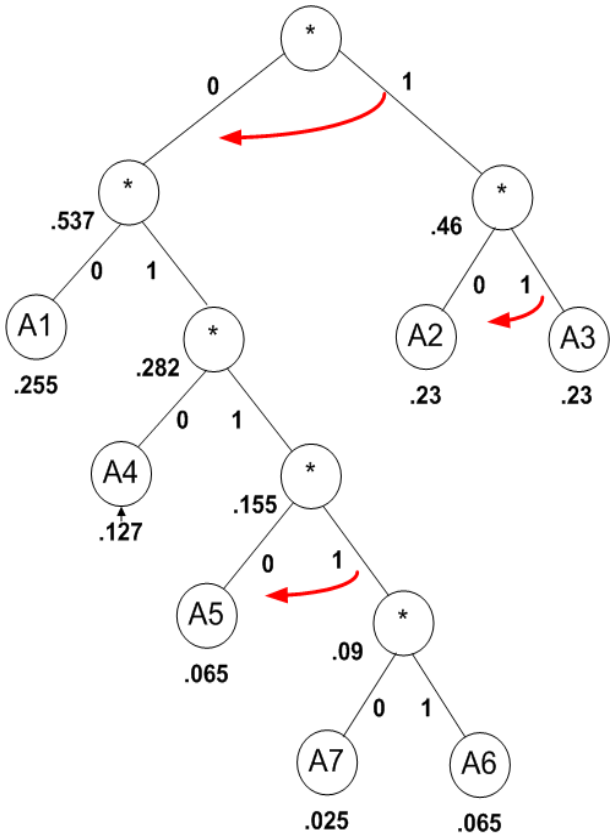


Figure 5. Example of Huffman Code Tree with 3 bits flipped.

bit error occurs to the one after which the correct parsing resumes. This concept is illustrated in Figure 6 with an error propagation of 4 codewords. In [16-20] the mean error propagation length (MEPL) of codeword trees have been broadly studied. The concept of MEPL is also referred to as expected error span by Maxted and Robinson in [14]. In [14], the authors studied an error transition model and gave a method for computing the MEPL. This model was further extended by Swaszek and DiCicco in [20]. The formulations in [14], [16], [20] are in algebraic forms and, as mentioned in [18], a symbolic algebraic software is necessary for computing MEPL, especially when the code size is large. In [21], Takishima *et al.* presented a formula for computing MEPL based on crossover probability, which was further simplified to a new theorem in [20]. We are using the theorem in [26] to calculate the MEPL used in our encryption technique.

Our goal is to breach the prefix condition and make it unfeasible to read or guess the current or subsequent codewords. By finding the MEPL value, we can approximate after how many codewords the decoder might be able to resynchronize. Thus we alter at least one bit every ¾*MEPL codewords and at the most one bit per codeword according to the user control security parameter *f* (as discussed in section 2.2), thus forcing the decoder to be always unsynchronized. The randomness of the bit flipping operation yielding scenario (1) and scenarios (2 and 3) discussed above adds a confusion feature to the encrypted bits. This confusion makes the encryption output uncorrelated within a subsequence of codewords.

## IV. SECURITY ANALYSIS

In this section we analyze the robustness of our scheme against different digital attacks, including ciphertext-only attack, known-plaintext attack, and chosen-plaintext attack. The analysis will also consider attacks for each part of our scheme: the random number generator and the shuffling/flipping bit operations. As mentioned earlier, we are using different random numbers for each shuffling/flipping operation, which eliminate the practicability of known-text attack and chosen-plaintext attack to find out the next shuffling or bit flipping operation.

### *2.1 CipherText-Only attack :*

Cipher Text-only (Brute force) attack can target the two main components of our encryption scheme: random number generator or the bit flipping/shuffling level. First we will analyze the complexity of brute force attack on our random number generator then the bit flipping/shuffling level:

#### *1) Chaotic generator*

**Key space analysis:** A good cryptosystem must be sensitive to its private key, and the key space must be large enough to make a brute force attack computationally an infeasible task. The secret key provided by the user in our scheme is assumed to be at least 128 bits, which is split into 2N chunks to initialize variables and parameters for the generation of the PRNG (eqs.1-5).

For a 128-bit Key (see Table II), we allow at least 8 bits to a maximum of 16 bits per parameter or variable in the initialization process, representing a network of 8 to 4 chaotic maps, respectively. The weights ε and $w$ are obtained by iterating one of the maps a random number of times. If an attacker decides to break the system key by brute force, it will need an order of $2^{128}$ operations (just the complexity of the private key $B \geq 128$ ); if the attacker rather decides to brute force guessing of the variables and parameters in an N-map network, it will need at least ($2^{32*N*2}$) operations (without considering $w_i$ and $\varepsilon$), which represents $2^{256}$ and $2^{512}$ operations for the minimum and maximum allowed number of maps (N), respectively.

#### *2) Bit flipping/shuffle*

Brute Force attack may include restoring the corrupted codewords. In this case the attacker needs to find the correct bit sequence and the correct boundary of the original stream codewords. For this to happen the attacker has to try all possible combination of bit flipping and shuffling to find the original codeword's; the complete decoded frame or motion is the only guarantee of the correct order of bits and segments.

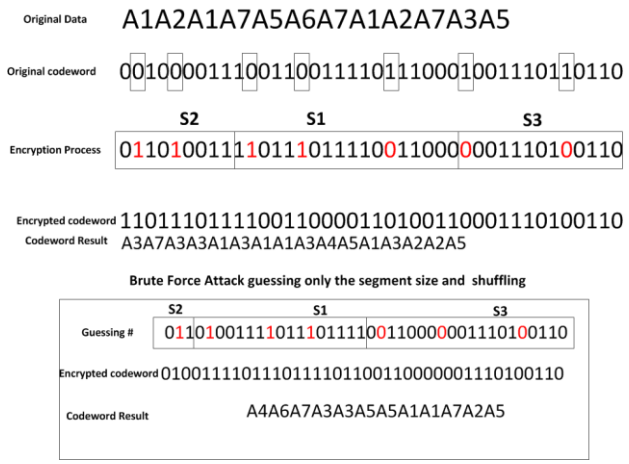


Figure 6. illustration of Error Propagation in Huffman codeword's

First we show the complexity to detect the location of the bit that is flipped and restore it back. Second we show the complexity to guess the shuffling order. To simplify the problem we assume that the attacker knows the average size of codewords, the MEPL value and the $f$ value. Therefore he knows the range of the bit location the system flips the bits within.

Let $V=\{v_1,v_2, \dots ,v_J\}$ be video stream of J bits, the system flips one bit every M<<J bits and break up the stream into R segments and then shuffles them. A brute force attack should perform both flipping and shuffling attack at the same time to decide which video stream is the valid one. In video compression, just decoding part of the image will not produce a valid output; the sequence of codewords is decidedly depending on previous and subsequent codewords in order to produce a valid image. Therefore, the bit flipping complexity attack for the encrypted stream for J/M segments of M bits can be expressed as:

$$O(Bits\ Flipping) = M^{J/M}$$

On the other hand, the segment shuffling complexity attack requires the trying of all possible segment positions:

$$O(\text{Segment Shuffling}) = R!$$

Hence the brute force complexity for both bit flipping and segment shuffling is:

$$O(N) = R!\, M^{J/M}$$

In video transmission, the size of bit streams J depends on the video properties. For example, for one I-Frame of 320*240 video decoded in MPEG4 format, the size will be 2Kb. The average calculated MEPL value for this codec is 3, and the

average codeword size is 6 bits. And let's assume that the value of R is 20. Then brute force attack complexity is:

$$O(N) = 20!\,6^{910} \cong 6^{950} \cong 2^{1630}$$

The complexity of constructing one frame of image using brute force is a very complex task. In addition to the computation complexity, the process of recognizing a valid video output while trying bit flipping and segment shuffling requires system training and consumes a lot of computation to make decision based on the training set. This is again an extremely complex task.

*2.2 Known-plaintext and Chosen-plaintext attacks:*

In these attacks, the only candidate part to be targeted is the PRNG (chaotic generator). The bit-flipping, segmentation, and shuffling blocks are vulnerable to these attacks, but only within the same RTP packet. Successfully decoding one packet has no bearings on the next packet, as it uses different setup (bit flipping, shuffling, and segment size). Consequently, the attacker has to break down the entire chaotic system (Eq.1).

If we assume that the attacker has found the partial values of the chaotic variable *Out* (least significant 27 bits used in the encryption process per map), it only knows partial information about the PRNG since *Out* is a mask of three chaotic trajectories from three different maps. So, the next step for breaking the system is to find the remaining 5 bits of *Out*, and solve for the Xi's and $\lambda_i$'s altogether. The attacker will need at least $2^{(N=\{4,8\})*(32+5)} = 2^{\{148,296\}}$ operations (32 bits for the parameter $\lambda$) to tear down the second security wall given that he knows the partial values of *Out* (this complexity is without considering $w_i$'s and $\varepsilon$ in Eq.1). If we take into account that each map en Eq.3 is randomly selected, the complexity of the attack increases considerably. The security of the scheme can be adjusted according to the secrecy of the ongoing multimedia communication, higher security requires increasing both the number of maps and/or the number of bits left out in the *Out* variable.

## V. PERFORMANCE EVALAUATION SETUP

To evaluate the performance and power efficiency of our encryption scheme and compare it with full encryption methods, we have implemented multiple schemes on the following computing platforms: Dell desktop, Lenovo laptop, Asus netbook, Nokia 900 and Nokia N800 PDA. Table III shows specifications of these platforms. These platforms are chosen to reflect the diversity in computation power and mobility characteristics. We evaluate and compare each scheme in terms of CPU usage, encryption speed and power consumption (where available).

We installed Ubuntu Linux distribution on all devices except Nokia N900 and N800 which runs Maemo 5 [12] distribution. Maemo is a mobile operating system for Nokia PDAs based on Debian GNU/Linux operating system.

TABLE III. SPECIFICATIONS OF DIFFERENT PLATFORMS USED IN EXPERIMENTS

| Testbed | CPU Type | Clock Speed | Memory | Operating System |
|---|---|---|---|---|
| Desktop | Intel Duo Core 2 | 2.2 Ghz | 3 GB | Ubuntu8.3 |
| Laptop | Intel Duo Core 2 | 2.2 Ghz | 2 GB | Ubuntu9.1 |
| NetBook | Intel Atom | 1.6 Ghz | 1 Gb | Ubuntu Netbook |
| Nokia N800 | TI Omap 2420 | 333 Mhz | 128 MB | Maemo |
| Nokia N900 | TI OMAP 3430 | 600 Mhz | 256 MB | Maemo 5 |

TABLE IV. FEATURES OF RECHARGEABLE BATTERIES USED IN EXPERIMENTS

| | **Laptop** | **Netbook** | **Nokia N900** |
|---|---|---|---|
| *Battery Capacity* | 4752 mAh | 4400 mAh | 1320 mAh |
| *Voltage* | 11100 mV | 11100 mV | 4400 mV |
| *Type* | Li-Ion | Li-Ion | Li-Ion |

Ubuntu implements battery management using the Advanced Configuration and Power Interface (ACPI), which exports battery data via the /proc/acpi/battery file system. The data values exported by ACPI expressed by millivolts and milliamps which can be converted to Watt; Given the current voltage and amps of battery using ACPI values, we compute energy consumed (in Watt Hour).

We conducted energy consumption tests on three battery equipped devices: laptop, netbook and N900, while we conducted encryption speed and CPU usage on all the four devices. Table IV shows battery specification of the laptop, netbook, N900 devices used in energy consumption tests. We adopted the following methodology to measure energy consumption. Each device is first fully charged and each encryption schemes is modified so that it runs in an infinite loop. For each scheme, we periodically polled (every 60 seconds) the Linux ACPI values and computed energy usage. To measure the actual energy used by encryption operation, each device is first fully charged and then left on idle running. We periodically polled (every 60 seconds) the Linux ACPI values and computed the energy consumption. The difference of the idle energy and energy consumed during encryption operation is reported as the energy consumed by the encryption scheme.

Full encryption schemes are focused on the AES Standard, which provides higher speed encryption than obsolete encryption standards like DES, RC, etc, without degrading the

encryption robustness. We compare our selective technique against full encryption schemes represented by different implementations of AES and a nonconventional encryption scheme based on chaos theory. We chose Bernstein [1], an AES implementation that takes advantage of the architecture-dependent reduction of instructions used to compute AES and the microarchitecture-dependent reduction of cycles used for those instructions. We also chose the earlier AES implementation by Gladman [4] and PolarSSL [18]. For chaotic Encryption we chose Hasimoto [5] scheme. We did not compare our scheme with other selective encryption schemes as they require modification of media encoders/decoders.

The values of f, Av, MEPL used on all experiments are 1/3, 6, 3 respectively. In other words, the previous values implements the maximum bit flipping of our scheme, that is 1 bit flipped per average codeword Av. Thus the reported results reflect the maximum complexity of our scheme. Our scheme is ~3 times faster when the values are adjusted to the minimum bit flipping case ($\frac{3}{4} \cdot Av \cdot MEPL$).

## VI. Experimental Results

In this section we first analyze the speed and CPU usage on each platform, followed by battery dissipation. Figure 7 compares the CPU usage assuming different streaming rates; this means that all implementations are processing the same amount of data over the same period of time. On the desktop, laptop and netbook the proposed scheme is using ~15% less CPU power than the chaotic scheme. But the selective encryption and chaotic encryption are using ~150% less CPU power than all other AES encryption schemes on laptop, desktop and netbook. On Nokia N900 and N800, the proposed scheme is using ~2 times less CPU than all other encryption schemes even the chaotic scheme whose CPU usage was close to our scheme. On netbook Gladman's scheme improved its performance in term of CPU usage. OpenSSL CPU usage performance was poor in all platforms.

As shown in the graphs of Figure 8, the selective encryption is by far the fastest scheme on all platforms, with an average of ~2 (~200%) times faster than the fastest AES implementation and 15% faster than the chaotic encryption scheme. Among the AES implementations, Bernstein's implementation obtained the best performance in term of speed, except on the netbook and the N800 device, where Gladman's was the fastest among all the other AES implantations. On N800, the proposed scheme performance was faster than all other Encryption scheme by a factor of ~3, and on N900, the proposed scheme enchanted its speed performance to be ~5 times faster than all other encryption schemes. This performance improvement is due to the inclusion of a math co-processor in the Nokia N900 architecture (needed for computing chaotic trajectories in Eq.1).

Figure 9 shows the battery energy consumption on battery operated devices. Figure 9a shows the performance of encryption schemes in terms of energy usage and the amount of data encrypted within 60 minutes running time. A linear behavior for all implementation is observed with AES implementation having the greatest slopes or energy consumption. For example, Bernstein's implementation encrypts 280 Gbytes of data and consumes over ~32 Watt Hour of battery energy on the laptop platform. On the other hand:

1- The Chaotic scheme encrypts 800 Gbytes of data and consumes the same ~32 Watt Hour of the energy on the laptop.

2- The Selective scheme encrypts almost 900 Gbytes of data and consumes the same ~32 Watt Hour of the energy on the laptop.

Figure 9b shows the energy consumed for different data sizes. In Overall, selective encryption can almost process ~200% - ~800% more information with the same energy consumption than all other scheme implementations on all platforms except for chaotic encryption on the laptop, desktop and netbook where the proposed scheme can process ~200% more information with the same energy consumption.

Figure 10 shows the loss of energy resulted by performing the maximum encryption speed of each scheme on the N900 device from full charge state until a complete discharge. As we see from Figure 9, all the schemes follow the same power loss pattern. But in the case of Gladmans scheme and the proposed scheme the battery lives longer even when both are operating in full speed like all others. The reason for this is the power control features of Nokia N900. These results shows that the encryption scheme can be further enhanced by controlling which processor units are used in the encryption.

In Table V we compare the proposed scheme with existing work in terms of other parameters and security features.

TABLE V. Comparison among different encryption schemes

| Scheme | Requires Decoding | Vulnerable to plaintext attacks | Affects Compression | Applicable at Inter-mediate network nodes |
|---|---|---|---|---|
| Shashank[1] | Yes | Yes | No | No |
| Meyer [2] | Yes | No | No | No |
| Spanos[3] | Yes | Yes | No | No |
| Tang[4] | Yes | No | Yes | No |
| Wu[6] | Yes | Yes | Yes | No |
| Wen[7] | Yes | Yes | Yes | No |
| Proposed Scheme | No | No | No | Yes |

## VII. CONCLUSION

In this paper we have presented a highly scalable encryption scheme for VLC multimedia bit streams that uses computationally efficient chaotic maps based method to generate random numbers. These secure random numbers are then utilized to diffuse correlations among codewords. The proposed scheme is highly robust and scales well in terms of encryption speed and CPU usage with the increase in streaming rate. Our implementation results show over 100% speedups in execution times across multiple platforms. In terms of CPU usage (and indirectly power usage), the proposed scheme is at least 100% across platforms. In the work presented in this paper, security is achieved partly due to a robust random number generator. In our future work, we plan to explore codec specific selective schemes and most robust shuffling and diffusion operations that may work on less secure random number generators.

## REFERENCES


[1] Bernstein, D.J., Schwabe, P. : New AES Software Speed Records. INDOCRYPT-2008, LNCS Vol. 5365, 322-336. (2008)

[2] Capocelli, R., Santis, A., Gargano, L., Vaccaro U.: On the construction of statistically synchronizable codes. IEEE Trans. on Information Theory, Vol., 407-414. (1992)

[3] Ferguson, T., Rabinowitz, J.: Self synchronization Huffman codes. IEEE Trans. On information theory, Vol. 30, 687-693. (1984)

[4] Gladman, B : AES and combined encryption /authentication modes. [Online]. Available: http://fp.gladman.plus.com/AES/ (2006)

[5] Hasimoto-Beltran, R.: High-performance multimedia encryption system based on chaos. Chaos 18, 023110. ( 2008)

[6] Hasimoto-Beltran, R,. Ramírez-Ramírez, R.: Cycle-Detection for Secure chaos-based encryption. Commun. Nonlinear Sci. Numer. Simulat., Vol. 16, 3203-3211. (2013)

[7] Hemami, S.: Robust image transmission using resynchronizing variable-length codes and error concealment. IEEE journal on selected areas in communicates, Vol. 18, 927-939. (2000)

[8] Huffman, D.A.: A Method for the Construction of Minimum-Redundancy Codes", Proceedings of the I.R.E., 1098–1102. Huffman's original article. (1952)

[9] Jakimoski, G., Subbalakshmi, K.: Cryptanalysis of some multimedia Encryption Schemes. IEEE Trans. On Multimedia, Vol. 10, 330-338. (2008)

[10] Khanvilkar, S., Khokhar, A. A.: Efficient transmission of MP3 streams over VPNs. GLOBECOM. (2006)

[11] Liu, X.,. Eskicioglu, A. M: Selective encryption of multimedia content in distribution networks: Challenges and new directions. 2nd Int. Conf. Communications, Internet, and Information Technology, Scottsdale, Az. (2003)

[12] Maemo. [Online]. Available: http://maemo.org/

[13] Masato, K., Satoshi, N.: Burst Error Recovery for Huffman Coding. IEICE Trans. Inf. Syst., Vol . l0. E88-D, 2197-2200. (2005)

[14] Maxted J., Robinson, J.: Error recovery for variable length codes. IEEE Trans. on Information Theory, Vol. 31, 94-801. (1985)

[15] Meyer, J., Gadegast, F.: Security Mechanisms for Multimedia Data with the Example MPEG-1 Video. [Online]. Available: http://www.gadegast.de/frank/doc/secmeng.pdf

[16] Monaco, M. E., Lawler, J. M.: Corrections and additions to ‘error recovery for variable length codes. IEEE Trans. Inform. Theory, vol. IT-33, pp. 454–456. (1987)

[17] Montgomery B., Abraham J.: Synchronization of binary source codes. IEEE Trans. on Information Theory, Vol. 32, 849-854. (1986)

[18] National Institute of Standards and Technology (NIST): FIPS-197: Advanced Encryption Standard (AES). [Online]. Availabe: http://www/csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[19] Spanos, G. A., Maples, T. B.: Performance Study of a Selective Encryption Scheme for the Security of Networked, Real-time Video. Proceedings of 4th International Conference on Computer Communications and Networks, Las Vegas, NV. (1995)

[20] Swaszek, P. F., DiCicco, P.: More on the error recovery for variable-length codes. IEEE Trans. Inform. Theory, vol. 41, 2064–2071. (1995)

[21] Takishima, Y. , Wada, M, Murakami, H. : Error states and synchronization recovery for variable length codes. IEEE Trans. Commun., vol. 42, 783–792. (1994)

[22] Tang, L.: Methods for Encrypting and Decrypting MPEG Video Data Efficiently. Proceedings of the 4th ACM International Multimedia Conference, Boston, MA, 219-230. (1996)

[23] Te-Chung, Y., Kumar, S. : A low complexity error recovery technique for wavelet image codecs with inter-subband dependency. IEEE Trans. On Consumer Electronics, Vol. 48, 973- 981. (2002)

[24] Wen, J., Kim, H., Villasenor, J. : Binary arithmetic coding with key-based interval splitting. IEEE signal process. Lett., Vol. 13, 69-72. (2006)

[25] Wu, C., Kuo C.:Design of integrated multimedia compression and encryption systems. IEEE trans. Multimedia, Vol. 7, 828-839. (2005)

[26] Zhou, G., Zhang, Z.: Synchronization Recovery of Variable-Length Codes. IEEE Trans. Inform. Theory, Vol. 48, No. 1, 219-227. (2002)

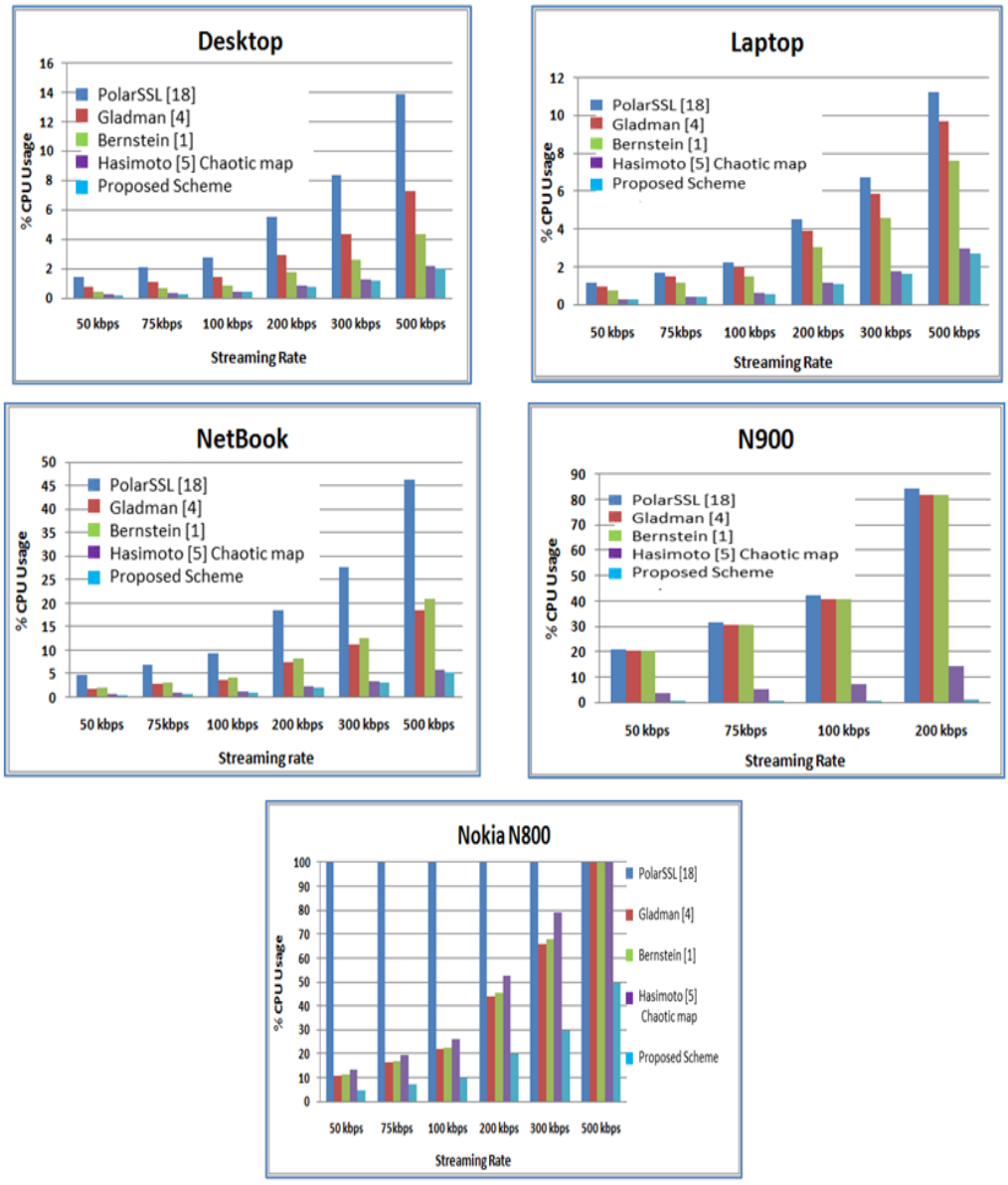


Figure 7. Performance results on different platforms in terms of encryption speed.

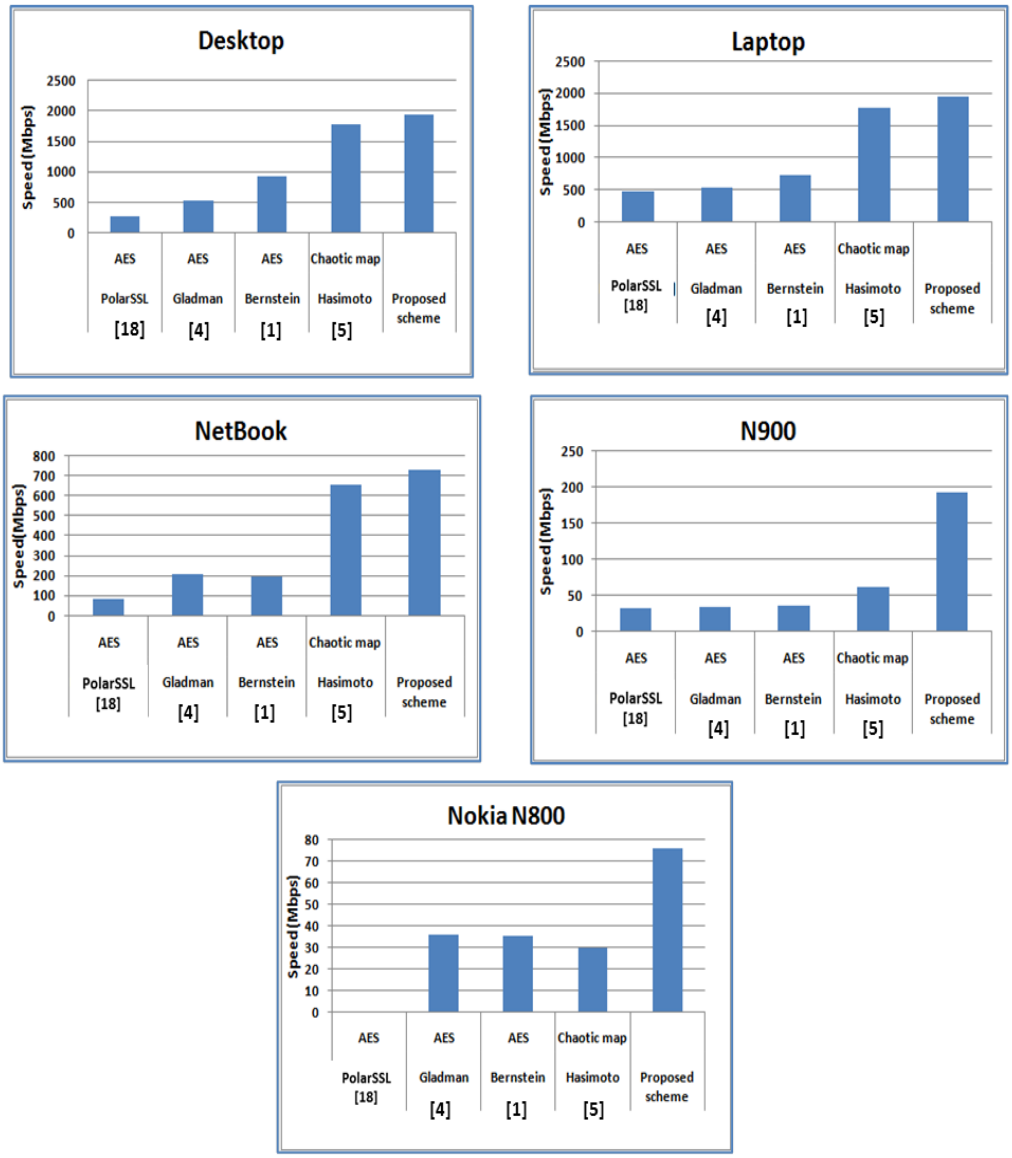


Figure 8. Performance results on different platforms in terms of CPU usage.

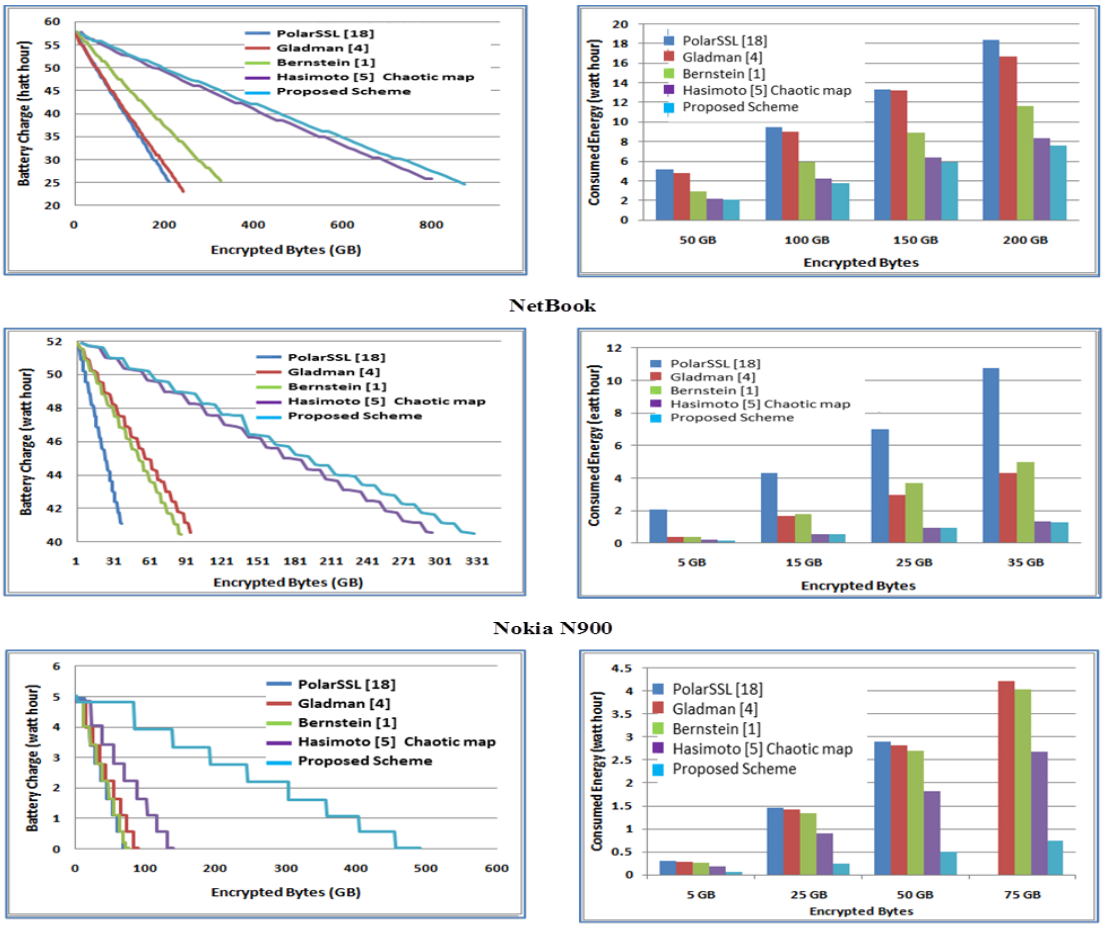


Figure 9. Performance results in terms of energy consumed

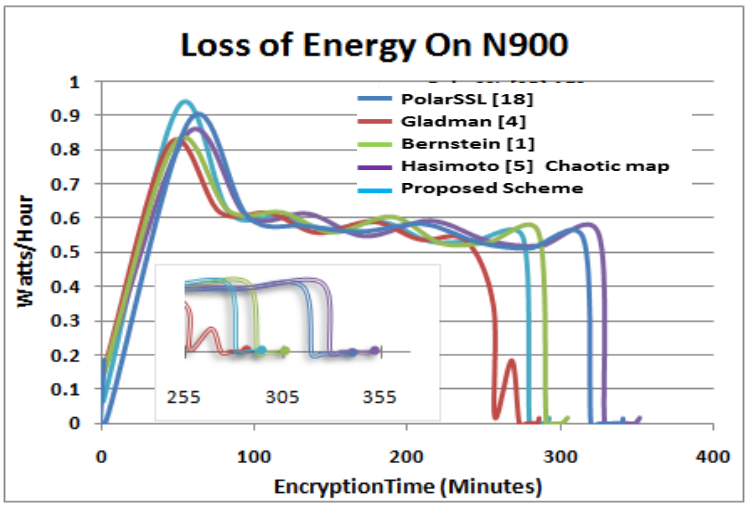


Figure 10. Loss of Energy on N900