*Entropy* **2014**, *16*, 5575-5600; doi:10.3390/e16105575

OPEN ACCESS

*entropy*

**ISSN 1099-4300**

www.mdpi.com/journal/entropy

*Article*

# Partial Encryption of Entropy-Coded Video Compression Using Coupled Chaotic Maps

**Fadi Almasalha [1], Rogelio Hasimoto-Beltran [2,*] and Ashfaq A. Khokhar [3]**

[1] Faculty of Information Technology, Applied Science Private University, Shafa Badran, 11931 Amman, Jordan; E-Mail: f_masalha@asu.edu.jo

[2] Center for Research in Mathematics (CIMAT), Apartado Postal 402, Guanajuato, C.P. 36000, México

[3] Department of Electrical and Computer Engineering, Illinois Institute of Technology, 3300 South Federal Street, Chicago, IL 60616-3793, USA; E-Mail: ashfaq@iit.edu

* Author to whom correspondence should be addressed; E-Mail: hasimoto@cimat.mx; Tel.: +52-473-7327155.

External Editor: Kevin H. Knuth

*Received: 9 August 2014; in revised form: 1 October 2014 / Accepted: 17 October 2014 / Published: 23 October 2014*

---

**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 from a secure and computationally efficient chaos-based pseudo-random number generator. The proposed scheme is ubiquitous to the end users and can be deployed at any node in the network. It provides different levels of security, with encrypted data volume fluctuating between 5.5–17%. It works on the compressed bit stream without requiring any decoding. It provides excellent encryption speeds on different platforms, including mobile devices. It is 200% faster and 150% more power efficient when compared with AES software-based full encryption schemes. Regarding security, the scheme is robust to well-known attacks in the literature, such as brute force and known/chosen plain text attacks.

**Keywords:** chaotic maps; Real-Time Protocol (RTP); video encoding; Huffman code

---

## 1. 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 the sheer volume of data involved, the temporally-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 a greater 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.

Numerous selective encryption schemes have been proposed for multimedia streaming applications (see [1] for an in-depth review). Following Liu and Koening [1], video encryption can be classified into two main categories; joint compression-encryption (within the encoding process) and compression-independent encryption, which can be performed before or after the encoding process. Joint compression-encryption algorithms are pervasive and codec dependent; they modify the standard video codec and may affect the final compression ratio (this is the most common form of encryption in the literature). On the other hand, compression-independent encryption, in particular the encryption before compression, is rarely used, because cryptographic operations eliminate redundancy, reducing considerably the compression ratio. Encryption after compression takes into account the relationship between different video elements (for example, Intra, Predicted and Bidirectional frames) or headers distribution to alter the VLC bit stream. We describe some of the most representative selective encryption schemes in the literature. Khanvilkar *et al*. [2] proposed a selective encryption approach for mp3 bitstreams that partially encrypted selected fields in the mp3 header. Meyer and Gadgast [3] proposed a selective encryption scheme for MPEG-1 bit streams. The principle data to be secured included: all the headers, Intra frames and Intra blocks. They proposed a number of combinations of the above scheme to attain different levels of security. Spanos and Maples [4] proposed encrypting the $I$ frames and the ISO start and end code of the MPEG stream. Tang [5] proposed an approach to use a random permutation list instead of the zigzag order for mapping an $8 \times 8$ DCT block to a $1 \times 64$ 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 Eskicioglu [6] 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) a decrease in the compression performance of entropy coding;
(3) insignificant computational reduction with respect to total encryption;
(4) a lack of bitstream compliance;
(5) an increase in key size;
(6) require compression decoding.

Wu and Kuo [7] 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*. [8] 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 [8] and [7] are vulnerable to low complexity known and/or chosen plain text attacks. Park and Shin [10] proposed a selective encryption scheme for hierarchical video encoding using different keys for each processed layer: the intra-prediction modes (base layer), the motion vector difference values and the sign bits of the texture data (enhancement layer). In Wang and Tian [11], the intra-prediction mode, motion vector difference and quantization coefficients are encrypted. They developed a hierarchical key generation method, in which the encryption keys are generated based on a cryptographic hash function. A more recent scheme proposed by Lui *et al*. [12], describes a format compliant encryption for the H.264 that modifies the Trailing_ones_sign_flag and level_suffix in the CAVLC (context-based adaptive variable length coding) mode for both I and P frames.

Independently of their classification, the encryption schemes mentioned above are codec dependent; they search for video elements in the variable length coded (VLC) bit stream. Our solution is a novel partial encryption technique aimed at minimizing the encryption complexity while preserving security. It provides the following advantages compared to previous work: (1) video codec independent; it is targeted to be used after compression to secure VLC bit streams (mp3, H.263, H.264, *etc*.) without searching for video elements, such as headers, motion vectors, transform coefficients, *etc*, and without affecting video compression; (2) it can be implemented at any stage of the communication pipeline, that is after the encoding process on the sending device or at a dedicated server handling multiple multimedia streams (impossible for current schemes in the literature); (3) scalable; the level of security (or volume data to be encrypted) can be tuned according to user or application requirements; (4) computer performance aware; it is flexible enough to be run under different CPU power capabilities going from high-end to low-end performance (mobile phones, netbooks, iPads, Laptops, *etc*.); (5) a new pseudo-random number generator (PRNG) based on coupled chaotic maps that provides long cycles and a uniform distribution for increased security; and (6) excellent performance (to the best of our knowledge, this is one of the fastest schemes in the literature). Because our scheme makes no distinction of video elements, the encryption of the VLC stream generates decoding errors that propagate as long as the resulting codewords are valid. The longer the propagation error, the more secure is the system.

Contrary to our aim of generating long propagation errors, several works have been proposed to minimize this effect on subsequent codewords. In [13–18], solutions have been investigated to overcome such errors by exploiting the 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 work involving self-synchronizing codes that quickly help reestablish synchronization and, thus, reduce the run length of error propagation can be found in [13–18]. A more complex solution is to approximate the corrupted coefficient value from previous or/and advance values. Te-Chung and Kumar [18] have studied the dependency in inter-sub-band coefficient values and showed that there exists certain correlation between the coefficient of parent and child sub-bands of a picture. Based on the sub-band coefficient relation, they have proposed a scheme to recover the lost values. The scheme divides the corrupted sub-band 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 bit errors for which the upcoming bit sequence can be resynchronized; not a threat in our encryption schemes, since we intensively flip bits in the entire VLC stream. For additional details on security techniques related to VLC bit streams, readers are referred to [16,19].

The main idea of our encryption scheme is to make the decoding/resynchronization of the VLC codewords in the bitstream computationally infeasible in the absence of a secret key. Assuming the video stream consists of packets, each packet is divided into random-sized segments. Within each segment, bits are randomly flipped, such that the correlation present among codewords is diffused. Then, all of 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 plain text attacks.

The rest of this paper is organized as follows: In the next section, we describe the major components of the proposed encryption scheme. Section 3 analyzes the 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. The conclusions of the work are presented in Section 7.

## 2. Proposed Scheme

The encryption process is applied after the RTP packetizing process (Figure 1). The RTP payload (VLC bit stream) is transformed by performing the following dynamic operations (they change for every received packet): random bit flipping, packet division into L segments and packet segment shuffling. The position of the bits to be flipped, segment size and segment shuffling depend on random values from our own secure chaotic-based pseudo-random number generator. A block diagram of the proposed scheme is given in Figure 2. It is composed of three main blocks: (1) secure random number generator based on coupled chaotic maps; (2) bit flipping; and (3) segment shuffling. Details for each block are discussed in the next subsections.

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

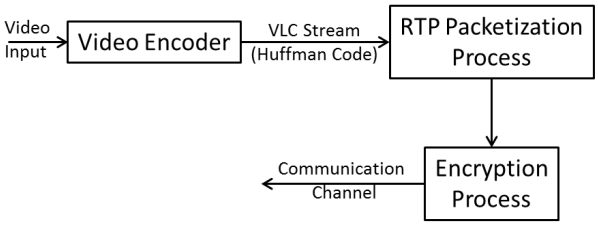
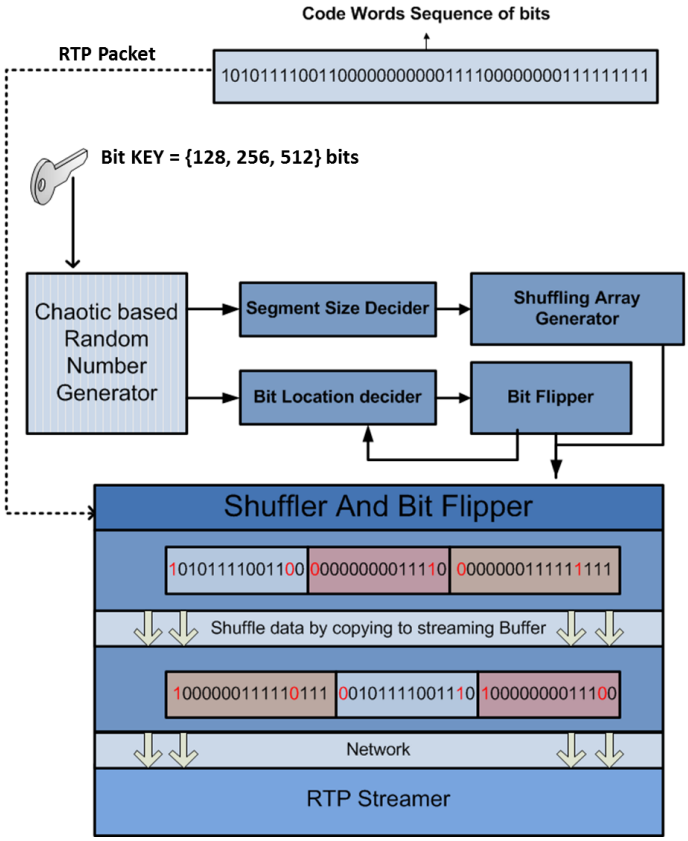



**Figure 2.** Illustration of the proposed encryption scheme.




### *2.1. Secure Random Number Generator Based on Chaotic Maps*

The security of the proposed scheme with respect 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 bit stream sequence (the attacker may know some parts of the bit stream without it). Current PRNGs 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 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 [3]. 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 ciphertext and plain text statistically independent, whereas diffusion is intended to spread out the influence of a single plain text digit over many ciphertext digits to hide the statistical structure of the plain text. 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 the trajectory, as performed in [3]. 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 [20]. In this work, we use CML to develop a PRNG that is robust to ciphertext, known/chosen plain text and differential attacks.

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

$$\begin{aligned} X_{i,j} &= (1-\varepsilon) f(X_{i,j-1}) + \varepsilon H(X_{i,j-1},\ \ldots\ , X_{N,j-1}) \\ H(X_{i,j-1},\ \ldots\ , X_{N,j-1}) &= \sum_{i=1}^{N} w_i X_{i,j-1}. \end{aligned} \tag{1}$$

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 $N \geq 8$. 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).

Equation (1) may be modified to include plain text feedback as part of the coupling function $H$, allowing diffusion of the information onto the entire ciphertext output:

$$H_j(X_{i,j},\ \ldots\ , X_{N,j}, P) = \sum_{l=1}^{N} w_l X_{i,j} + w_{N+1} P_{prev} \tag{2}$$

where $P_{prev}$ represents the sum of all previous plain text that were ciphered in the lifetime of $H_{j-1}$. 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 $\varepsilon$. Even though the computational complexity of the scheme is slightly increased, a single plain text change produces a totally different ciphertext, therefore increasing its robustness to statistical attacks.

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

$$X_{i,j} = f(X_{i,j-1}) = \lambda X_{i,j-1}(1 - X_{i,j-1}) \qquad \lambda \in [1,4) \qquad X \in (0,1) \tag{3}$$

where $\lambda$ represents the chaotic parameter and $X$ the state variable. To keep the good chaotic properties of the logistic map, we avoid bad initial values of $X$ and $\lambda$ and endorse the use $\geq 8$ chaotic maps in order to increase the cycle length period. A viable alternative for extending the cycle-length of a chaotic system is the use of cycle tracking schemes, as the one proposed in [21]. Once a cycle is detected, the system is perturbed to modify future trajectories and extend the cycle length without affecting its speed

performance. It is important to point out that any chaotic map in the literature (Renyi map, piece-wise linear map, *etc*.) can be used in Equation (1). The security relies on the scheme itself, rather than the chaotic map used in the scheme.

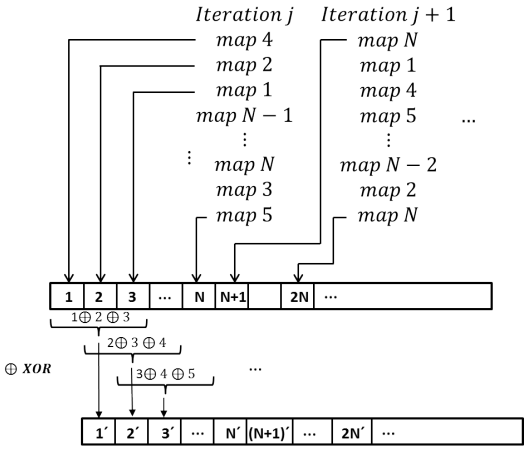
As mentioned before, the main reason for developing our own PRNG scheme is to manage the security of the system by creating a (near) cycle-free chaotic signal capable of handling long-term multimedia communications, such as VoIP and video streaming. The security is controlled by changing the number of chaotic maps and periodically perturbing the system state (variables, parameters, coupling function, weight $\varepsilon$, *etc*).

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

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

(2) A moving XOR window of a size of three is applied to the entire generated chaotic sequence generated by Equation (1), as shown at the bottom of Figure 3. This operation along with the random map selection modify the distribution generated by Equation (1) to yield a more robust quasi-uniform distribution for the ciphering process, as shown in Figure 4.

**Figure 3.** Representation of random map selection and the moving XOR window applied to chaotic trajectories.

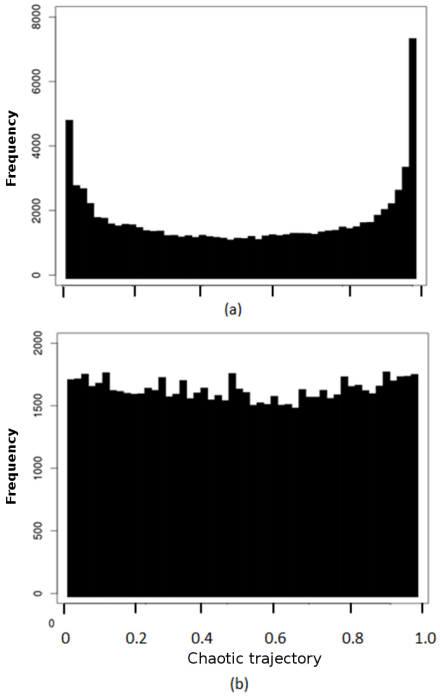



(3) The actual random number to be used in the encryption process is produced by operating three different maps according to the following formula:

$$Out_i = X'_{i,j} \tag{4}$$

where $X'_{i,j} = XI_{i,j} \oplus XI_{i+1,j} \oplus XI_{i+2,j}$, which comes from XORing the integer representation ($XI$) of three randomly chosen maps from Equation (1) (see action 2), and $\oplus$ is the XOR operator.

**Figure 4.** (**a**) Frequency distribution of chaotic trajectories generated by Equation (1). (**b**) Same as (a), with random map selection and a moving XOR window.




(4) Only $1 \leq U \leq 27$ 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 4). 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.

### 2.1.1. Chaotic System Initialization

The proposed scheme is symmetric; therefore, the initial system-key $(K)$ of size $B$ bits for $B \geq 128$ 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 as follows (see Figure 5):

$$X_{i,0} = K_{n/2}(2i-1)/2^{n/2},$$

$$\lambda_i = 3.68 + \frac{[K_{n/2}(2i)/2^{n/2} + K_{n/2}(2i)10^{h_{n/2}} + (a \oplus b)/2^{n/4}]}{10} \cdot \frac{[0.3187]}{MAX}, \tag{5}$$

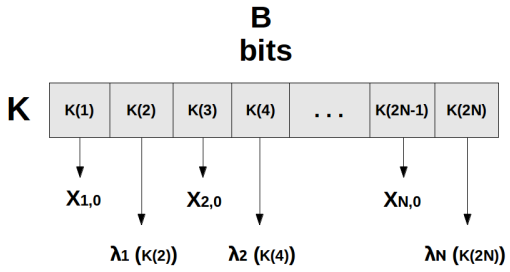
$$i = 1, 2, \ \ldots \ , N$$

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 $N \ll B$ and $(B \ mod \ N) = 0$, $K_{n/2}(m)$ 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, the $a \oplus b$ 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 $3.68 \leq \lambda_i \leq 3.998$ for $\lambda_i \neq \lambda_j$ and $i \neq j$, respectively. The remaining variables in Equation (1) $\varepsilon$ and $w_i$, $1 \leq i \leq N$ 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 $w_i = X_{p,i} / \sum_{j=1}^{N} X_{p,j}$, 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.

**Figure 5.** System-key $(K)$ partition $(n/2)$ for the creation of maps variables $(X_i, 0)$ and corresponding parameters $(\lambda_i)$.




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 a 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 Equation (5), as follows:

$$\begin{aligned} N &= [KS \ mod \ (MAX_B + 1)], \\ KS &= \textstyle\sum_{i=1}^{D/8} KEY_8(i) \end{aligned} \tag{6}$$

where $MIN_B \leq N \leq MAX_B$, following Table 1.

**Table 1.** 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** $(\mathrm{MIN_B})$ | **Maximum Number of Maps** $(\mathrm{MAX_B})$ |
|---|---|---|
| 128 | 8 | 16 |
| 256 | 8 | 25 |
| 512 | 16 | 32 |

### *2.2. Bit Flipping*

Once the encryption system receives an RTP packet (as shown in Figure 2), the main step in our scheme is to diffuse and destroy the meaning of the compressed codeword sequence and make it impractical to predict the original codeword sequence, as explained in Section 3. 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 3), $Av \cdot MEPL$ is the average error propagation in bits, and $f$ is a tunable security factor with values $1/(MEPL) \leq f \leq 1$. For this range of $f$, $Av \leq BF \leq (Av \cdot MEPL)$, that is at least one bit is changed per average Huffman codeword size $(Av)$ up to $(Av \cdot MEPL)$ bits. This is what makes our system scalably secure: 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 $B_i$ of the bit to be flipped in the stream is calculated as follows:

$$B_i = [rnd(\lceil \log_2(f \cdot Av \cdot MEPL) \rceil) \; mod \; BF] + 1 \tag{7}$$

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:

$BF = f \cdot Av \cdot MEPL$
for $i = 0$ to $(Ps * 8)/BF$ steps do
  $B_i = [rnd(\lceil \log_2(BF) \rceil) \; mod \; BF] + 1$
  $loc = i * (BF)$
  $Flip(P[B_i + loc])$

where $(Ps * 8)/BF$ is the number of bits to be flipped in the packet and $B_i$ is the position of the $t - th$ bit to be flipped in current block $loc = i * (BF)$ with length $BF$ (there may be many blocks or flipped bits per packet segment).

### *2.3. Segment Shuffling*

After bit-flipping, the RTP-packet payload is permuted by performing an L-way shuffling process. Here, the payload is divided into $20 \leq L \leq 65$ segments, which are shuffled using the algorithm described below and illustrated in Figure 6. The value of $L$ represents a security-control variable of the shuffling process, which is indirectly set by users. The specified user-defined levels of security are low, medium and high, representing $20 \leq L \leq 35$, $36 \leq L \leq 50$ and $51 \leq L \leq 65$ segments, respectively. The higher the number of segments $L$ in the RTP packet, the higher the security level, since brute force complexity to de-shuffle the packet is $L!$. In terms of power of two $2^S$ (for $S$ integer), the brute force complexity to de-shuffle the packet is the maximum $S$ for which $L! \geq 2^S$ for $60 \leq S \leq 272$ and $20 \leq L \leq 65$.

Once the shuffling security level is specified, $L$ becomes a random variable obtained as follows:

$$L = (rnd(\lceil \log_2(L_{\max}) \rceil)\ mod\ (L_{\max} - L_{\min})) + L_{\min} \tag{8}$$

where $L_{\min}$ and $L_{\max}$ integers are the limits of $L$ in the corresponding security level (low, medium or high). In the case of medium security $L_{\min} = 36$ and $L_{\max} = 50$; therefore, Equation (8) yields a random number between $36 \leq L \leq 50$. The number of segments is computed for every RTP packet.

Once $L$ has been computed, the segment size $Sg$ is calculated using Equation (9):

$$Sg = \frac{Ps}{L} \tag{9}$$

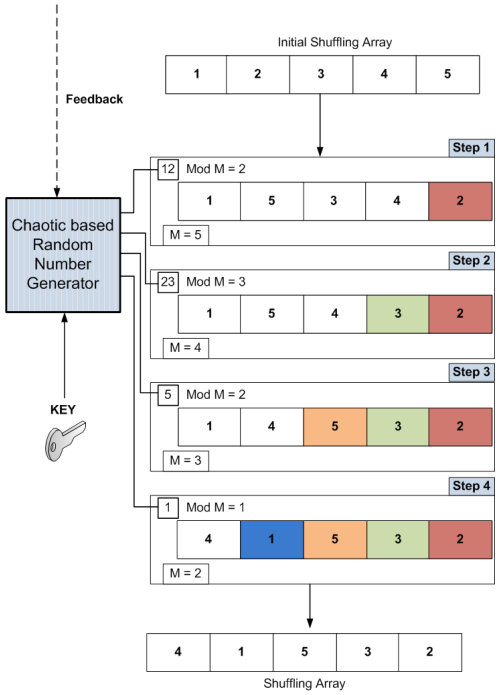
where $Ps$ is the RTP payload size in bytes. For a 300-byte RTP packet, the segment size randomly falls between $5 \leq Sg \leq 15$, which corresponds to $20 \leq L \leq 60$ segments.

The proposed shuffling mechanism starts by creating the shuffling array $A$ of size $L$ that will be used to shuffle the segments in $\mathcal{O}(L!)$ steps, as illustrated in Figure 6. Each entry in the initial shuffling array is the original block index. The scheme will iterate $L - 1$ times, starting by the initial value of $M = L$. The following algorithm generates the shuffling array:

Initialize $A$
$M = L$
for $L - 1$ steps do
  Generate random number $R$
  $T = R\ mod\ M$
  Swap($A[T], A[M]$)
  $M = M - 1$

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.

**Figure 6.** Illustration of building the shuffling array for an RTP packet with $L = 5$ segments.



## 3. Bit Flipping System Analysis

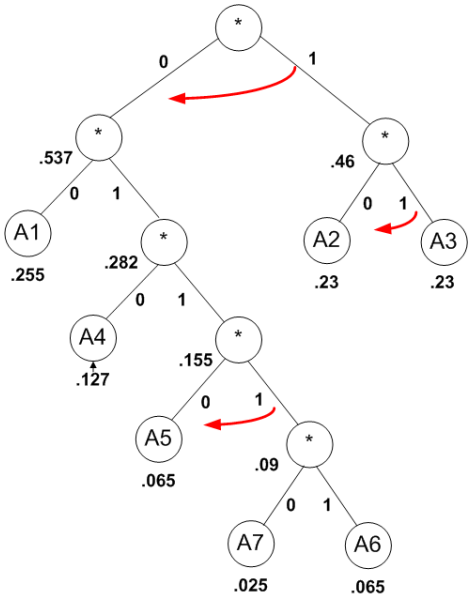
The key feature in our encryption is to make the reading and decoding of Huffman codes impossible without knowing the private key. The key characteristic exploited here is the prefix condition of the Huffman codes, which states that no code words can be a prefix of another code word [22].

The prefix code condition is used to guarantee the unique parsing path for each codeword, as shown in Figure 7. This condition provides hidden (implicit) borders that separate codewords from each other. As we see in Figure 7, 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 the red arrow in Figure 7, then a transition will occur at that point, which results in 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 of the flipped bit for either a valid or invalid codeword. This is unlikely to happen in VLC streams, 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, that is the DC or AC coefficient, motion vector, header, *etc*. For instance, if the codeword holds the DC value of a block (smallest data-unit for video encoding corresponding to $8 \times 8$ pixels), the block will not be decoded correctly, as well as all subsequent blocks (until a synchronization marker is found). However, if the codeword holds the AC value of the block, only that block will not be decoded correctly (unless the block is used in motion compensation). The most critical scenario is when a header codeword is modified, which impedes the (total or partial) progression of the video decoding or playback.

**Figure 7.** Example of a Huffman code tree with three bits flipped.




The second and third scenarios will break the hidden borders between codewords, thus making the reading process misaligned and incorrect, as shown in Figure 8. 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 2.2) 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 bit error occurs to the one after which the correct parsing resumes. This concept is illustrated in Figure 8 with an error propagation of 11 codewords. A visual example of error propagation is depicted in Figure 9. Flipping one bit in the football encoded video sequence (in Byte 5000) caused severe damage during video playback for about three seconds. After this time, the system either resynchronized itself or a SYN marker was found. In [4,17,23,24], 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 [16]. In [16], 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 [24]. The formulations in [16,23,24] are in algebraic forms and, as mentioned in [25], a symbolic algebraic software is necessary for computing $MEPL$, especially when the code size is large. In [26], Takishima *et al.* presented a formula for computing $MEPL$ based on crossover probability, which was further simplified to a new theorem in [24]. We are using the theorem in [27] to calculate the $MEPL$ used in our encryption technique.

**Figure 8.** Illustration of error propagation in Huffman codewords.

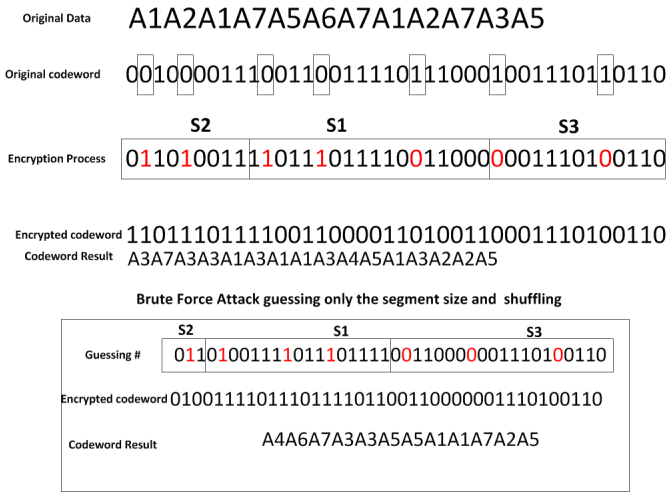



**Figure 9.** Error propagation effect caused by flipping one bit in the encoded football video sequence. Original (**left**) and corrupted images (**right**) are shown.

Our goal is to breach the prefix condition and make it infeasible to read or guess the current or subsequent codewords. By using the $MEPL$ value, we can approximate the number of codewords the decoder might be able to resynchronize. Thus, we alter at least one bit every $MEPL$ 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.

## 4. Security Analysis

In this section, we analyze the robustness of our scheme against different digital attacks, including ciphertext only attack, known plain text attack and chosen plain text 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 plain text attack to find out the next shuffling or bit flipping operation.

### *4.1. Ciphertext-Only Attack*

Ciphertext-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.

#### 4.1.1. 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 (Equations (1)–(5)).

**Table 2.** Specifications of different platforms used in the experiments.

| Testbed | CPU Type | Clock Speed | Memory | Operating System |
|---|---|---|---|---|
| *Desktop* | Intel Duo Core 2 | 2.2 Ghz | 3 GB | Ubuntu 8.3 |
| *Laptop* | Intel Duo Core 2 | 2.2 Ghz | 2 GB | Ubuntu 9.1 |
| *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 |

For a 128-bit key (see Table 2), we allow at least eight bits to a maximum of 16 bits per parameter or variable in the initialization process, representing a network of eight to four chaotic maps, respectively. The weights $\varepsilon$ 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 $2^B$ operations (just the complexity of the private key for $B \geq 128$); if the attacker rather decides to brute force guessing of the variables and parameters in an $N$-map network, it will need at least $(2^{32\times N\times 2})$ 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.

#### 4.1.2. 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 codewords; the complete decoded frame or motion is the only guarantee of the correct order of bits and segments.

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 locations that the system flips the bits within the RTP packet.

Let $V = \{v_1, v_2, \ldots, v_J\}$ be the entire video stream of $J$ bits. The system flips one bit every $M \ll J$ bits (for $(M \leq BF)$ and breaks up the stream into $R$ segments, 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 dependent 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:

$$\mathcal{O}(Bits\ Flipping) = M^{J/M} \tag{10}$$

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

$$\mathcal{O}(Segment\ Shuffling) = R! \tag{11}$$

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

$$\mathcal{O}(N) = R!M^{J/M} \tag{12}$$

In video transmission, the size of bit streams $J$ depends on the video properties. For example, for one $I$-Frame of $320 \times 240$ video decoded in MPEG4 format, the size will be 2 Kb. The average calculated $MEPL$ value for this codec is three, and the average codeword size is six bits. Additionally, let us assume that the value of $R$ is 20. Then, brute force attack complexity is:

$$\mathcal{O}(N) = 20!6^{910} \cong 6^{950} \cong 2^{1630} \tag{13}$$

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 a decision based on the training set. This is again an extremely complex task.

### *4.2. Known Plain Text and Chosen Plain Text 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 a different setup (bit flipping, shuffling and segment size). Consequently, the attacker has to break down the entire chaotic system (Equation (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. Therefore, the next step for breaking the system is to find the remaining five bits of $Out$ and to solve for the $X_i$'s and $\lambda_i$'s altogether. The attacker will need at least $2^{(N=\{4,8\})\times(32+5)} = 2^{\{148,296\}}$ 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 Equation (1)). If we take into account that each map in Equation (1) is randomly selected and transformed with a size-three XOR window, the complexity of the attack increases considerably. The security of the scheme can be adjusted according to the secrecy of the ongoing multimedia communication, and higher security requires increasing both the number of maps and/or the number of bits left out in the $Out$ variable.

## 5. Performance Evaluation Setup

To evaluate the performance and power efficiency of our encryption scheme and to 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 2 shows the 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 a Ubuntu Linux distribution on all devices, except the Nokia N900 and N800, which run a Maemo 5 [28] distribution. Maemo is a mobile operating system for Nokia PDAs based on the Debian GNU/Linux operating system.

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 are expressed by millivolts and milliamps, which can be converted to watts; given the current voltage and amps of the battery using ACPI values, we compute the energy consumed (in W·h).

We conducted energy consumption tests on three battery-equipped devices, laptop, netbook and N900, while we conducted encryption speed and CPU usage on all four devices. Table 3 shows the battery specification of the laptop, netbook and N900 devices used in the 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 s) the Linux ACPI values and computed energy usage. To measure the actual energy used by the encryption operation, each device is first fully charged and then left to run idly.

We periodically polled (every 60 s) the Linux ACPI values and computed the energy consumption. The difference of the idle energy and the energy consumed during the encryption operation is reported as the energy consumed by the encryption scheme.

**Table 3.** Features of the rechargeable batteries used in the experiments.

| | **Laptop** | **Netbook** | **Nokia N900** |
|---|---|---|---|
| *Battery Capacity* | 4752 mAh | 4400 mAh | 1320 mAh |
| *Voltage* | 11,100 mV | 11,100 mV | 4400 mV |
| *Type* | Li-Ion | Li-Ion | Li-Ion |

Full encryption schemes are focused on the AES standard, which provides higher speed encryption than obsolete encryption standards, like Data Encryption Standard (DES), Rivest Cipher (RC), *etc.*, without degrading the encryption robustness. We compare our selective technique against full encryption schemes represented by different implementations of AES and a non-conventional encryption scheme based on chaos theory. We chose Bernstein [29], 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 [30] and PolarSSL [25]. For chaotic encryption, we chose the Hasimoto [20] scheme. We did not compare our scheme with other selective encryption schemes, as they require the modification of media encoders/decoders.

The values of $f$, $Av$ and $MEPL$ used in all experiments are 1/3, 6 and 3 respectively. These values reflect the maximum complexity or maximum bit flipping frequency of our scheme, that is one bit-flipped per average codeword $Av = 6$ bits. The minimum complexity or minimum bit flipping frequency is reached when $f = 1$, corresponding to one bit-flipped every $18$ bits.

## 6. Numerical Results

In this section, we analyzed the encryption scheme sensitivity to initial conditions, encryption space and performance on different platforms (see Table 2) according to the following metrics: speed, CPU usage and battery dissipation (only for mobile devices).

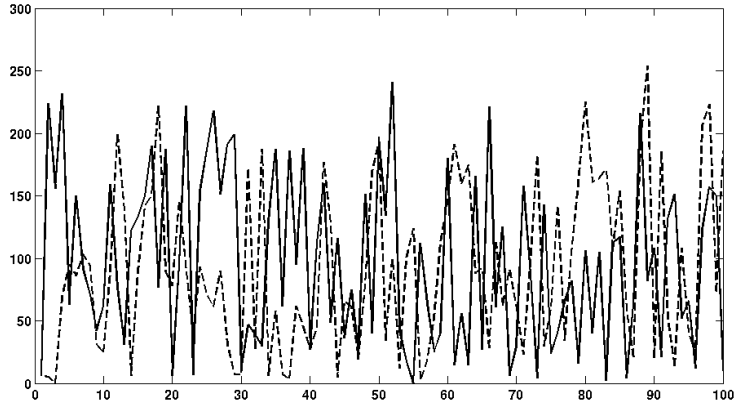
### *6.1. Sensitivity to Initial Conditions and Encryption Space*

When encrypting VLC compressed bit streams, it is difficult to perceptually analyzed the ciphertext, because coders are not able to successfully decode the modified image or video information. Headers, motion vectors, DC and AC coefficients, *etc*, are modified in such a way that decoders may not find known codewords in the bit stream. In order to provide a look at the effect of our encryption scheme, we substitute the compressed plain text for its non-compressed version. Our scheme transforms the original low-frequency image content to an irregular noisy look pattern, as depicted in Figure 10a,b respectively. This result gives us a clue of the effect of the bit-flipping scheme on compressed plain text, which is even worse due to its sensitivity to bit streams changes, as seen in Figure 9.

An important property of chaos-based encryption is sensitivity to initial conditions. A slight bit-change in the system key must produce totally different and uncorrelated transformations in both ciphertext and plain text. To test the sensitivity of our scheme to bit changes in the system key, we ciphered a non-compressed video frame (compressed video cannot be played back due to errors in the headers, motion vectors, *etc*.) with key K1 and deciphered the sequence with key K2, such that K1 and K2 differ in the least significant bit. The result of this experiment is shown in Figure 10. Figure 10a,b shows the original non-compressed video frame and corresponding ciphertext using key K1, respectively. Figure 10c shows the deciphered frame of Figure 10b using the same system key, K1; as expected, the original frame is perfectly reconstructed. Figure 10d shows the deciphered frame of Figure 10b using key K2. The effect of modifying the least significant bit in the system key impedes the reconstruction of the original image/video data, such that the resulting plain text is uncorrelated to the original plain text sequence (with a noise-like pattern). To evaluate the sensitivity of the scheme to plain text changes, we modified the least significant bit in the first byte of the non-compressed frame and obtained the ciphertext for both frames, the original (Figure 10b) and modified one (Figure 10e). Figure 11 represents the first 100 bytes of both cipher frames. As can be seen, the ciphertext trajectories have an independent random-like behavior with average correlation within $\pm 0.1$ (very low correlation for partially encrypted frames).

**Figure 10.** Sensitivity of the proposed scheme to a bit-change in the system key and plain text. (**a**) Original non-compressed image; (**b**) ciphertext; (**c**) deciphered image (same as the original); (**d**) same as (c) with the least significant nit (LSB) change in the system key; and (**e**) same as (d) with the LSB change in the first byte of plain text.


**Figure 11.** Plain text sensitivity to bit changes. Solid and dashed trajectories belong to the same plain text with one bit changed in the first byte.



In selective encryption, it is important that the analysis of the encryption space (volume of encrypted data) is evaluated as the percentage of the number of encrypted bits in the original video sequence. Since our scheme is scalable, for $\frac{1}{3} \leq f \leq 1$, the total volume of securely encrypted data fluctuates between 5.5–17%. Table 4 (taken from Wang *et al.* [11]) gives the comparison of different selective encryption algorithms in encrypted information, encryption operation and encrypted data volumes. We have included two additional schemes in the original table (in bold format), our scheme and Lui and Wong's scheme [12]. The minimum security level ($f = 1$) provided by our scheme represents less data volume securely encrypted than the other schemes.

**Table 4.** Comparison of the encrypted data volumes (reproduced from [11]) of several selective encryption schemes.

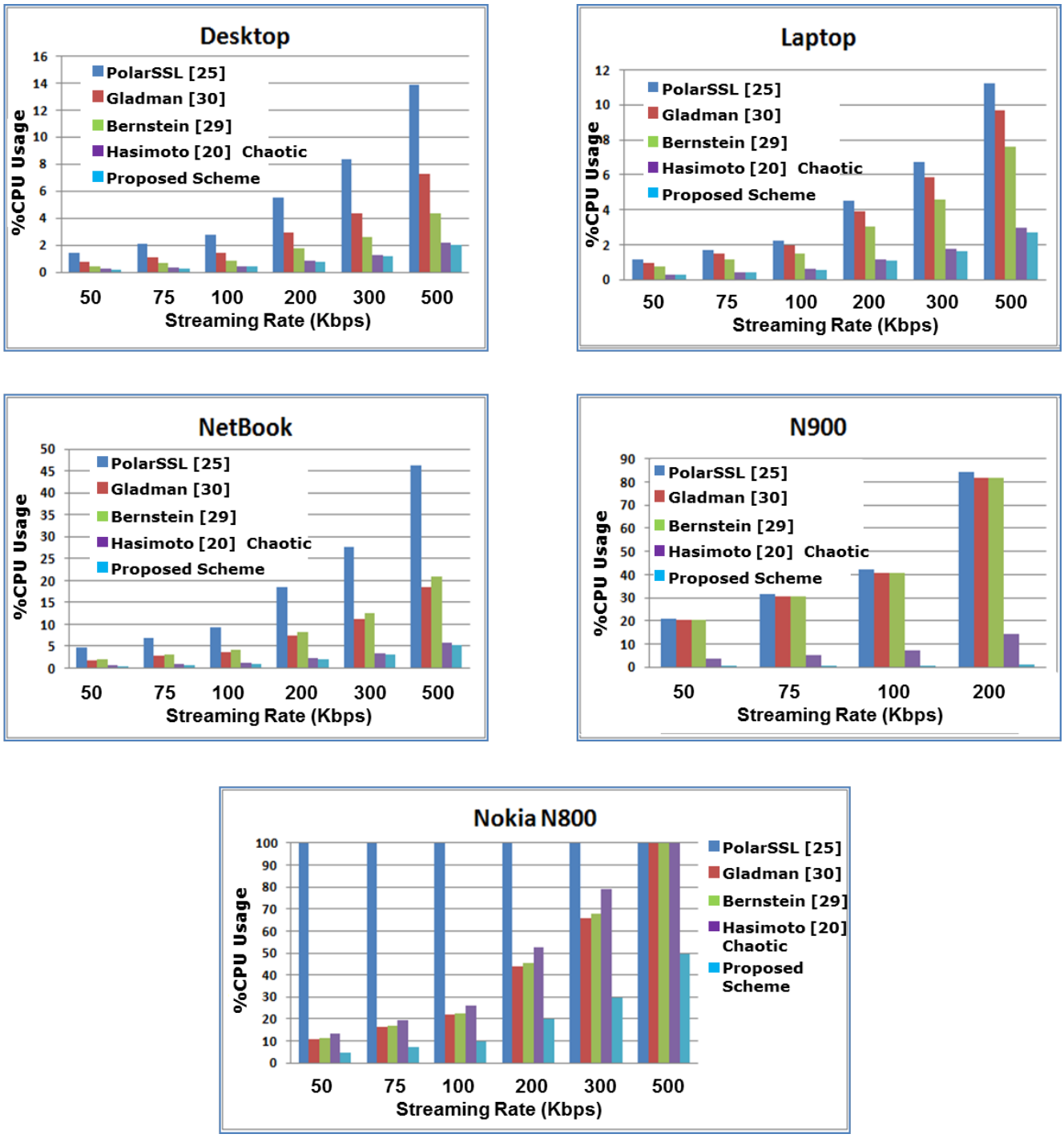
| Encryption Scheme for H.264 | Encrypted Information | Encryption Operation | Encrypted Data Volumes |
|---|---|---|---|
| Lui and Wong [12] | Syntax elements of the context-based adaptive variable length coding (CAVLC) | Chaos encryption | 19% |
| Lian *et al.* [31] | Intra-prediction mode, motion vector difference, residue data of the macro-blocks | Stream cipher or AES | 10% |
| Li *et al.* [32] | Intra-prediction mode, inter-prediction mode, transform coefficients, motion vectors | Scrambling, XOR operation, sign inversing | 9.8% |
| Jiang *et al.* [33] | Intra-prediction mode, inter-prediction mode, transform coefficients, motion vectors | Chaos encryption, XOR operation | 9.8% |
| Spinsante *et al.* [34] | Quantization parameter, intra-prediction mode, deblocking filter coefficients | Linear-Filter Shift Register (LFSR), XOR operation | 12.3% |
| Lian *et al.* [35] | Intra-prediction mode, motion vector difference, intra-macro-block's residue | Stream cipher | 10% |
| Wang *et al.* [11] | Intra-prediction mode, motion vector difference, quantization coefficients | XOR operation | 6.8% |
| Yuan *et al.* [36] | The sign bit of DC coefficient in I-frame, I-block in P-frame, I-block in B-frame | Chaos encryption | 6.2% |
| Our scheme | VLC error diffusion (bit-flipping) and shuffling | Chaotic encryption based on coupled chaotic maps | 5.5%–17% |

### *6.2. Encryption Scheme Performance*

In this section, our proposed encryption scheme performance on different platforms (see Table 2) is analyzed according to the following metrics: speed, CPU usage and battery dissipation (only for mobile devices). Figure 12 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 $\approx$ 15% less CPU power than the chaotic scheme [16]. However, the selective encryption and chaotic encryption are using $\approx$150% less CPU power than all

other AES encryption schemes on the laptop, desktop and netbook. On the 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 the netbook, Gladman's scheme improved its performance in terms of CPU usage. PolarSSL CPU usage performance was poor for all platforms.

**Figure 12.** Performance results on different platforms in terms of CPU usage.




As shown in the graphs of Figure 13, the selective encryption is by far the fastest scheme for all platforms, with an average ≈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 of the other AES implantations. On the N800, the proposed scheme performance was faster than all other encryption schemes by a factor of ≈3, and on N900, the proposed scheme enhanced its speed performance to be ≈50-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 Equation (1)).

**Figure 13.** Performance results for different platforms in terms of encryption speed.

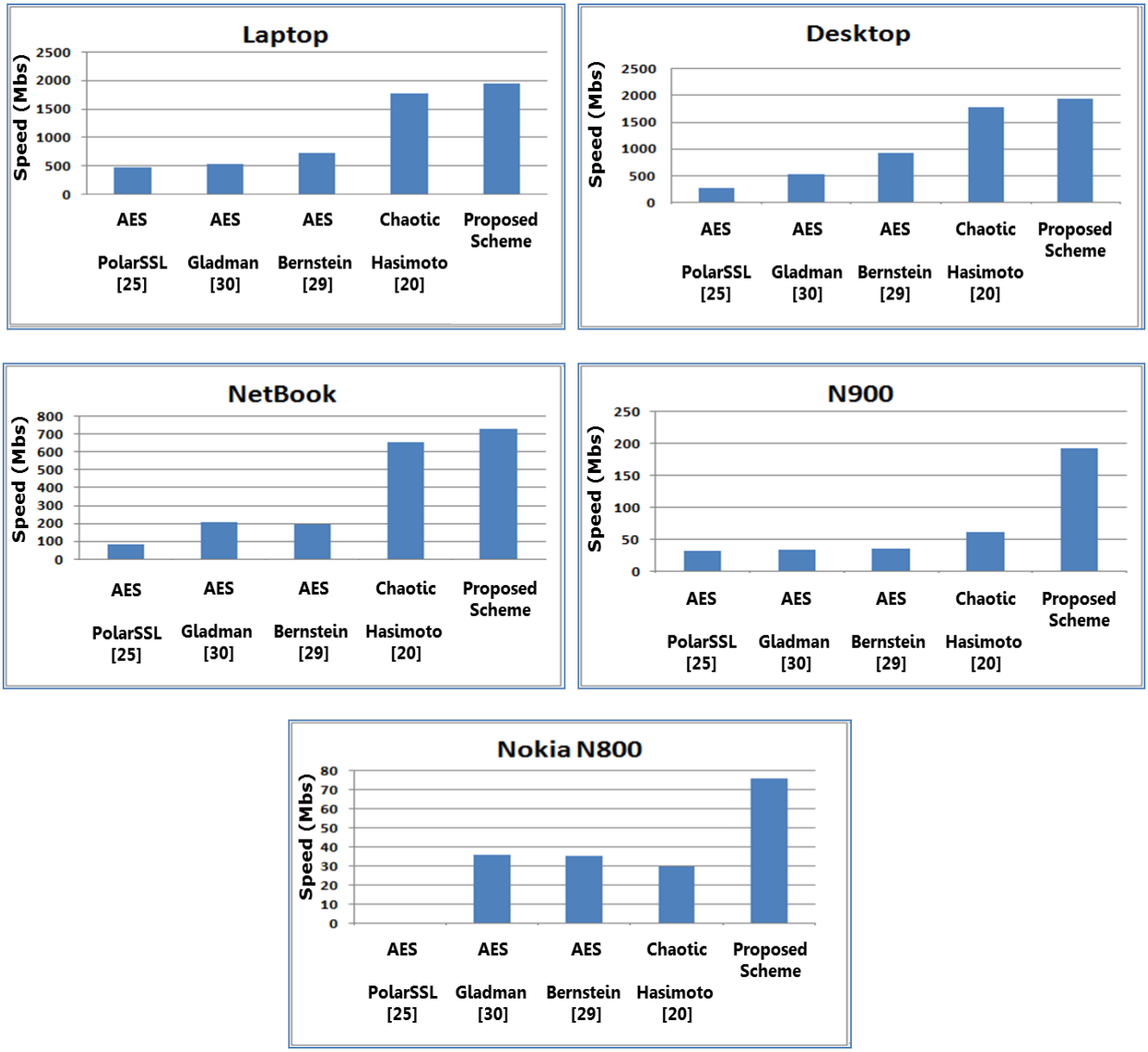



Figure 14 shows the battery energy consumption on battery-operated devices. Figure 14a shows the performance of encryption schemes in terms of energy usage and the amount of data encrypted within 60 min of running time. A linear behavior for all implementation is observed, with the AES implementation having the greatest slopes or energy consumption. For example, Bernstein's implementation encrypts 280 GB of data and consumes over ≈ 32 W·h of battery energy on the laptop platform. On the other hand:

(1) the chaotic scheme encrypts 800 GB of data and consumes the same ≈32 W·h of the energy on the laptop;

(2) The selective scheme encrypts almost 900 GB of data and consumes the same ≈32 W·h of the energy on the laptop.

Figure 14b shows the energy consumed for different data sizes. Overall, selective encryption can almost process ≈200%–≈800% more information with the same energy consumption as 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 14.** Performance results in terms of energy consumed.

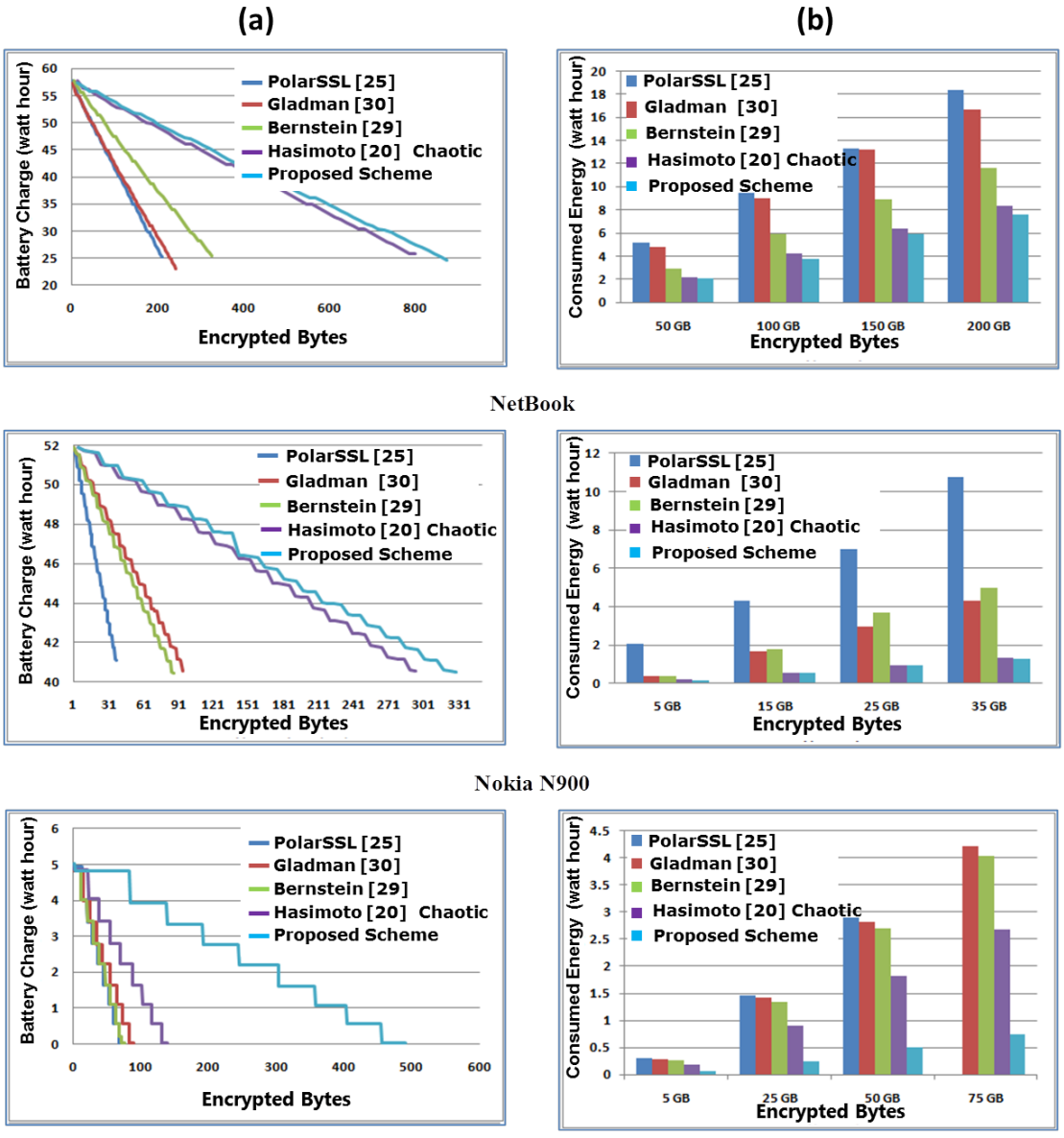
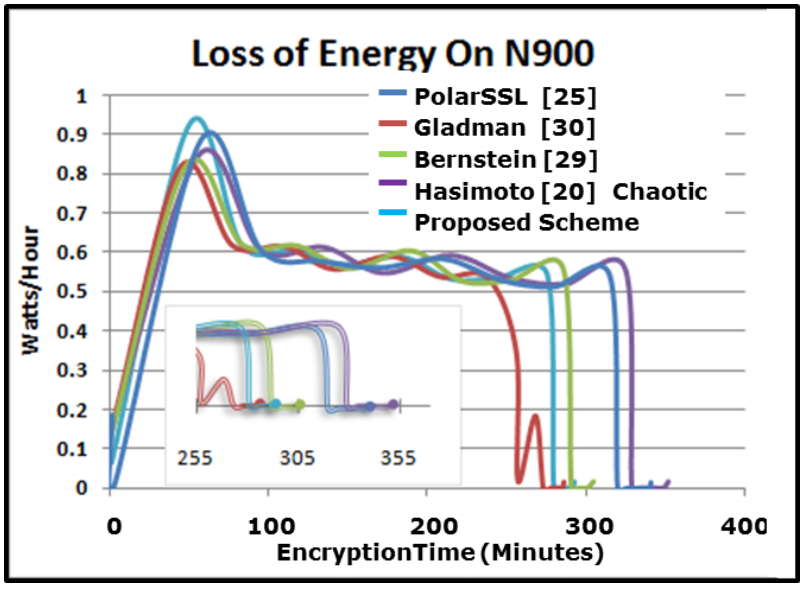



Figure 15 shows the loss of energy resulting from performing the maximum encryption speed of each scheme on the N900 device from the full charge state until a complete discharge. As we see from Figure 14, all of the schemes follow the same power loss pattern. However, in the case of Gladman's scheme and the proposed scheme, the battery lives longer, even when both are operating in full speed, as for all others. The reason for this is the power control features of the Nokia N900. These results

shows that the encryption scheme can be further enhanced by controlling which processor units are used in the encryption.

**Figure 15.** Loss of energy on the N900.




In Table 5, we compare the proposed scheme with existing work in terms of other parameters and security features.

**Table 5.** Comparison among different encryption schemes.

| Scheme | Requires Decoding | Vulnerable to plain text Attacks | Affects Compression | Applicable at Intermediate Network Nodes |
|---|---|---|---|---|
| Shashank [2] | Yes | Yes | No | No |
| Meyer [3] | Yes | No | No | No |
| Spanos [4] | Yes | Yes | No | No |
| Tang [5] | Yes | No | Yes | No |
| Wu [7] | Yes | Yes | Yes | No |
| Wen [8] | Yes | Yes | Yes | No |
| Proposed Scheme | No | No | No | Yes |

## 7. Conclusions

In this paper, we have presented a highly scalable encryption scheme for VLC multimedia bit streams that uses a 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 the most robust shuffling and diffusion operations that may work on less secure random number generators.

## Author Contributions

The work presented here is based on a collaborative effort of the authors. Each author contributed equally to different aspect of the publication. All authors have read and approved the final manuscript.

## Conflicts of Interest

The authors declare no conflict of interest.

## References

1. Lui, F.; Koening, H. A Survey of Video Encryption Algorithms. *Comput. Secur.* **2010**, *29*, 3–15.
2. Khanvilkar, S.; Khokhar, A. NIS07-1: Efficient Transmission of MP3 Streams over VPNs. In Proceedings of the GLOBECOM-06, IEEE Global Telecommunications Conference, San Francisco, CA, USA, 27 November–1 December 2006; pp. 1–5.
3. Meyer, J.; Gadegast, F. *Security Mechanisms for Multimedia Data with the Example MPEG-1 Video*; Project Description of SECMPEG; Technical University of Berlin: Berlin, Germany, 1995.
4. Spanos, G.A.; Maples, T.B. Performance Study of a Selective Encryption Scheme for the Security of Networked, Real-Time Video. In Proceedings of the Fourth International Conference on Computer Communications and Networks, Las Vegas, NV, USA, 20–23 September 1995; pp. 2–10.
5. Tang, L. Methods for Encrypting and Decrypting MPEG Video Data Efficiently. In Proceedings of the 4th ACM International Multimedia Conference, Wuhan, China, 9–11 September 1996; pp. 219–230.
6. Liu, X.; Eskicioglu, A.M. Selective Encryption of Multimedia Content in Distribution Networks: Challenges and New Directions. In Proceedings of IASTED Communications, Internet & Information Technology (CIIT), Scottsdale, AZ, USA, 17–19 November 2003; pp. 527–533.
7. Wu, C.; Kuo, C. Design of Integrated Multimedia Compression and Encryption Systems. *IEEE Trans. Multimed.* **2005**, *7*, 828–839.
8. Wen, J.; Kim, H.; Villasenor, J.D. Binary Arithmetic Coding with Key-Based Interval Splitting. *IEEE Signal Process. Lett.* **2006**, *13*, 69–72.
9. Jakimoski, G.; Subbalakshmi, K.P. Cryptanalysis of Some Multimedia Encryption Schemes. *IEEE Trans. Multimed.* **2008**, *10*, 330–338.
10. Park, S.; Shin, S. Efficient Selective Encryption Scheme for the H.264/Scalable Video Coding (SVC). In Proceedings of the Fourth International Conference on Networked Computing and Advanced Information Management, 2008, NCM '08, Gyeongju, Korea, 2–4 September 2008; Volume 1, pp. 371–376.
11. Wang, X.; Zheng, N.; Tian, L. Hash key-based video encryption scheme for H.264/AVC. *Signal Process. Image Commun.* **2010**, *25*, 427–437.

12. Lui, O.; Wong, K. Chaos-based selective encryption for H.264/AVC. *J. Syst. Softw.* **2013**, *86*, 3183–3192.
13. Capocelli, R.M.; Santis, A.A.D.; Gargano, L.; Vaccaro, U. On the Construction of Statistically Synchronizable Codes. *IEEE Trans. Inf. Theory* **1992**, *38*, 407–414.
14. Ferguson, T.; Rabinowitz, J. Self-Synchronizing Huffman Codes. *IEEE Trans. Inf. Theory* **1984**, *30*, 687–693.
15. Hemami, S.S. Robust Image Transmission Using Resynchronizing Variable-Length Codes and Error Concealment. *IEEE J. Sel. Areas Commun.* **2000**, *18*, 927–939.
16. Maxted, J.; Robinson, J. Error Recovery for Variable Length Codes. *IEEE Trans. Inf. Theory* **1985**, *31*, 794–801.
17. Montgomery, B.L.; Abrahams, J. Synchronization of Binary Source Codes. *IEEE Trans. Inf. Theory* **1986**, *32*, 849–854.
18. Te-Chung, Y.; Kumar, S. A Low Complexity Error Recovery Technique for Wavelet Image Codecs with Inter-Subband Dependency. *IEEE Trans. Consum. Electron.* **2002**, *48*, 973–981.
19. Kitakami, M.; Nakamura, S. Burst Error Recovery for Huffman Coding. *IEICE Trans.* **2005**, *88-D*, 2197–2200.
20. Hasimoto-Beltran, R. High-Performance Multimedia Encryption System Based on Chaos. *Chaos* **2008**, *18*, doi:10.1063/1.2903758.
21. Hasimoto-Beltran, R.; Ramirez-Ramirez, R. Cycle Detection for Secure Chaos-Based Encryption. *Commun. Nonlinear Sci. Numer. Simul. (CNSNS)* **2013**, *16*, 3203–3211.
22. Huffman, D.A. A Method for the Construction of Minimum-Redundancy Codes. *Proc. I.R.E.* **1952**, *40*, 1098–1102.
23. Monaco, M.; Lawler, J. Corrections and Additions to 'Error Recovery for Variable Length Codes' by J. C. Maxted and J. P. Robinson. *IEEE Trans. Inf. Theory* **1987**, *33*, 454–456.
24. Swaszek, P.; DiCicco, P. More on the Error Recovery for Variable-Length Codes. *IEEE Trans. Inf. Theory* **1995**, *41*, 2064–2071.
25. *FIPS-197: Advanced Encryption Standard (AES)*; National Institute of Standards and Technology (NIST): Springfield, VA, USA, 2001.
26. Takishima, Y.; Wada, M.; Murakami, H. Error States and Synchronization Recovery for Variable Length Codes. *IEEE Trans. Commun.* **1994**, *42*, 783–792.
27. Zhou, G.; Zhang, Z. Synchronization Recovery of Variable-Length Codes. *IEEE Trans. Inf. Theory* **2002**, *48*, 219–227.
28. The Home Of The Maemo Community. Available online: http://maemo.org/intro/ (accessed on 17 October 2014).
29. Bernstein, D.J.; Schwabe, P. New AES Software Speed Records. In *Progress in Cryptology—INDOCRYPT 2008*, Proceedings of 9th International Conference on Cryptology in India, Kharagpur, India, 14–17 December 2008; Chowdhury, D.R., Rijmen, V., Das, A., Eds.; Lecture Notes in Computer Science, Volume 5365; Springer: Berlin/Heidelberg, Germany, 2008; pp. 322–336.
30. Gladman, B. AES and Combined Encryption/Authentication Modes. Available online: http://www.gladman.me.uk/AES (accessed on 20 October 2014).

31. Lian, S.; Sun, J.; Liu, G.; Wang, Z. Efficient video encryption scheme based on advanced video coding. *Multimed. Tools Appl.* **2008**, *38*, 75–89.
32. Li, Y.; Liang, L.; Su, Z.; Jiang, J. A New Video Encryption Algorithm for H.264. In Proceedings of the IEEE 5th International Conference on Information, Communications, and Signal Processing, Bangkok, Thailand, 6–9 December 2005; pp. 1121–1124.
33. Jiang, J.; Li, Y.; Liang, L. Research and Improvement of the Video Encryption Algorithm for H.264. *Acta Electron. Sin.* **2007**, *9*, 1725–1726.
34. Spinsante, S.; Chiaraluce, F.; Gambi, E. Masking Video Information by Partial Encryption of H.264/AVC coding parameters. In Proceedings of the 13th European Signal Processing Conference, Antalya, Turkey, 4–8 September 2005.
35. Lian, S.; Liu, Z.; Ren, Z.; Wang, Z. Selective video encryption based on advanced video coding. In *Advances in Multimedia Information Processing-PCM 2005*, Proceedings of the 6th Pacific Rim Conference on Multimedia, Part II, Jeju Island, Korea, 13–16 November 2005; Ho,Y.-S., Kim, H.-J., Eds; Lecture Notes in Computer Science, Volume 3768; Springer: Berlin/Heidelberg, Germany, 2005; pp. 281–290.
36. Yuan, C.; Zhong, Y.; He, Y. Selective video stream encryption algorithm based on chaos. *Chin. J. Comput.* **2004**, *27*, 257–263.