Hindawi Publishing Corporation
EURASIP Journal on Information Security
Volume 2007, Article ID 35262, 18 pages
doi:10.1155/2007/35262

Research Article

Multimedia Encryption with Joint Randomized Entropy Coding and Rotation in Partitioned Bitstream

Dahua Xie and C.-C. Jay Kuo

Ming Hsieh Department of Electrical Engineering and Integrated Media Systems Center, University of Southern California, Los Angeles, CA 90089-2564, USA

Correspondence should be addressed to Dahua Xie, dahuaxie@gmail.com

Received 4 March 2007; Revised 21 July 2007; Accepted 11 September 2007

Recommended by E. Magli

This work investigates the problem of efficient multimedia data encryption. A novel methodology is proposed to achieve encryption by controlling certain operations in the data compression process using a secret key. The new encryption approach consists of two cascaded modules. The first one is called *randomized entropy coding* (REC) while the second one is called *rotation in partitioned bitstream* (RPB). By leveraging the structure of the entropy coder, the joint REC/RPB encryption scheme incurs extremely low computational and implementation costs. Security analysis shows that the proposed scheme can withstand the ciphertext-only attack as well as the known/chosen plaintext attack. The efficiency and security of the proposed encryption scheme makes it an ideal choice in secure media applications where a large amount of multimedia data has to be encrypted/decrypted in real time.

1. INTRODUCTION

The wide availability of digital multimedia contents as well as the accelerated growth of wired and wireless communication technologies have brought the multimedia content security issue to the forefront. In particular, the problem of efficient multimedia data encryption has recently gained more attention in both academia and industry. Although encrypting the entire multimedia content by a traditional cryptographic cipher (e.g., the block or stream cipher) yields a satisfactory level of security, such an approach does have several shortcomings. First, the computational cost associated with encrypting the entire multimedia content is often high due to the large data size. Second, the encryption and decryption operations add another level of complexity to the system. In most cases, additional hardware or software functions are needed in order to implement it. This is particularly unfavorable in certain applications such as mobile communications and embedded systems, where devices (e.g., cellular phones and portable equipments) are resource constrained due to the size limitation and the power consumption consideration. Hence, it is desirable to develop an efficient yet secure multimedia encryption technique.

In this work, the problem of multimedia encryption is investigated from a new angle. After a careful comparison between the multimedia compression process and the encryption process from the viewpoint of information theory, we point out that both can be in general viewed as a process to remove redundancy contained in the input. The key distinction between the two is that a secret key controls operations in encryption while all operations in compression are performed according to some standards. Based on this observation, a novel multimedia encryption methodology is proposed, where encryption is achieved by maneuvering certain operations in the compression process under the control of a secret key. Our new encryption approach consists of two stages. The first stage is called *randomized entropy coding* (REC). The core idea of REC is to use multiple entropy coding parameters/settings according to a random sequence inside the entropy coder. The second one is called *rotation in partitioned bitstream* (RPB), which further performs a random rotation to the output of the REC stage to yield the final bitstream.

This joint REC/RPB encryption paradigm has several advantages. First, the design leverages the structure of the entropy coder, thus demanding a negligible cost to implement in hardware or software. Second, encryption does not impair the compression ratio in the sense that the size of the encrypted bitstream is exactly the same as that obtained by standard compression. In terms of security strength, our proposed scheme can withstand various types of attack. The key space of a brute-force attack is studied and shown to be an exponential function of the plaintext/ciphertext length, which guarantees security under the ciphertext-only attack. Furthermore, we demonstrate how the REC/RPB cascade structure enhances the security by thwarting certain attacks. An interesting concept regarding the RPB encryption called the *equivalent key* is also developed. That is, there exist multiple different keys that can encipher the same plaintext to the same ciphertext. The properties of equivalent keys are studied and it is revealed that the average number of equivalent keys grows exponentially with respect to the size of the plaintext/ciphertext. This fact, combined with the cascade structure of REC/RPB encryption, provides strong resistance against the known/chosen plaintext attack.

The rest of this paper is organized as follows. Section 2 provides a brief overview of previous research work in this field. In Section 3, we compare the differences between compression and encryption, and propose a new multimedia encryption methodology of adding randomness into the compression process. The basic idea and the detailed implementation of the REC encryption scheme are presented in Section 3. Section 4 presents the RPB encryption scheme and investigates its key space and equivalent key properties. The computational cost is analyzed in Section 6. Section 7 examines the security strength of the joint REC/RPB encryption model with respect to the ciphertext-only attack and the known/chosen plaintext attack. The impact of RPB on the statistical randomness of input and output bitstreams is also discussed. Experiments are conducted to demonstrate the performance of the joint REC/RPB encryption scheme in Section 8. Finally, concluding remarks are given in Section 9.

2. PREVIOUS WORK

Encrypting the entire multimedia content imposes a heavy computational burden due to the large data size. Several selective encryption schemes have been proposed as a possible solution, where only a specific portion of the multimedia data is selected for encryption. In this section, we briefly review encryption schemes for DCT-based compression standards (e.g., JPEG, MPEG, H.264, etc.), which are widely used today. For wavelet-based and quadtree-based compression methods, we refer interested readers to [1–4] for more details.

Most existing selective encryption schemes are based on the encryption and/or scrambling of DCT coefficients and motion vectors, since it is generally believed that DCT coefficients and motion vectors carry more important semantic information. An encryption scheme called “Aegis” was developed by Spanos and Maples [5], which encrypts all the I frames of an MPEG video stream. Conceptually, B and P frames cannot be correctly decoded without the corresponding I frames. However, Agi and Gong [6] showed that a considerable amount of video contents is still visible largely due to unencrypted I macroblocks in B and P frames as well as the interframe correlation. Tang [7] proposed to encrypt DC coefficients by DES and use a random permutation (instead of the standard zigzag scan order) to scramble AC coefficients. Shi and Bhargava [8] proposed the video encrytion algorithm (VEA), where 64 most significant sign bits of DCT coefficients and motion vectors in each 16×16 macroblock are encrypted by a symmetric key cipher. An improved scheme called RVEA was presented in [9]. However, it was observed in [10] that even the DC and the first 8 AC coefficients are discarded for all DCT blocks, the reconstructed image still contains some meaningful content. Another scheme, called SECMPEG and developed by Meyer and Gadegast [11], provides four levels of security using a combination of selective encryption and additional headers. However, the system is incompatible with the standard MPEG encoder and decoder due to additional headers.

Qiao and Nahrstedt [12] proposed a scheme to split a bitstream into two halves odd and even according to a random pattern. The ciphertext c is obtained by the following operation

$$c = \text{odd} \oplus \text{even}, \tag{1}$$

where $\oplus$ is the XOR operation. The ciphertext c is then sent together with $E(\text{even})$, and E is an encryption cipher. Although this scheme cuts the encryption cost by half, it costs an additional step to recover the original data by assembling decrypted odd and even according to the random pattern. An even faster algorithm called the permutation encryption was proposed by Chu et al. [13]. It treats a bitstream in the unit of bytes and performs a byte permutation according to a key. The permutation operation yields a faster speed since it is much simpler than cryptographic operations. It is however a fixed byte-level permutation, which is shown to be vulnerable to the known plaintext attack in [14].

In summary, selective encryption schemes either incur a large computational overhead to achieve high security or fail to provide enough protection against attacks at a relatively low computational cost as compared to that of total encryption.

The recent trend in multimedia encryption research has placed more attention on integrating encryption with compression. Wu and Kuo [10, 15, 16] pioneered in this direction and proposed the use of multiple Huffman tables (MHT) alternately in a secret order in an entropy coder. Xie and Kuo [17] proposed an efficient encryption scheme for arithmetic coding by randomly alternating between two coding conventions in 2004. A very similar algorithm was later presented by Grangetto et al. in [18]. More recently, the use of key-based interval splitting to implement encryption in arithmetic coding was considered by Wen et al. [19, 20]. The work in [20] added an additional input permutation and output permutation on top of the scheme proposed in [19] in an attempt to enhance the security. Bose and Pathak [21] suggested an encryption scheme using a variable model arithmetic coder and the coupled chaotic system. Another encryption approach by random rotation in partitioned bitstreams was investigated by Xie and Kuo [22].

These papers have demonstrated promising results in integrating compression and encryption to achieve computational efficiency. However, some weakness of these schemes under advanced attacks has been pointed out by cryptanalysis. For instance, a recent study by Zhou et al. [23] revealed the weak key problem for the MHT scheme under some chosen plaintext attack. Thus, the design of an efficient and secure multimedia encryption scheme remains a challenging problem.

3. JOINT REC/RPB ENCRYPTION PARADIGM

The main deficiency of aforementioned encryption schemes is that they neglect one fundamental characteristic of coded multimedia data; that is, the compressed multimedia bitstream usually contains little redundancy as compared to regular data to be encrypted, for example, text documents and database files. This serves as an important basis in developing our new encryption methodology. By exploiting this feature, we design effective encryption schemes that can achieve high security strength at a relatively low computation cost. To better understand the interplay among redundancy, computation complexity, and security, we examine the operations of compression and encryption and make a comparison between these two.

Basically, encryption is a process of transforming an input (plaintext) that has a certain structure and semantics (meaning) to an output (ciphertext) that is statistically random and has no apparent structure. Under the control of a secret key, many rounds of complicated operations are performed to scramble the plaintext so as to produce the final ciphertext. These operations include logic operations (e.g., bitwise AND, OR, XOR, shift), mathematical operations (e.g., vector and matrix multiplications), and permutation and substitution, and so forth. As a result, the structure of the input file is completely scrambled without revealing any redundancy. The output appears to be a set of random data without any meaning. Thus, from the viewpoint of information theory, encryption can be considered as a transformation that hides redundancy contained in the input to produce a random output that is almost redundancy free.

Conceptually, a multimedia compression system works in a very similar fashion. Here, the input is the raw multimedia content (mostly video and audio) that contains a large amount of redundancy and the output is again an almost redundancy-free bitstream. Various compression techniques such as motion estimation, DCT transform, quantization, and entropy coding are exploited to remove rich redundancy in the raw content. The significant difference between encryption and compression is that operations in encryption are controlled by a secret key so that it is impossible to decrypt the original plaintext without knowing the key. While in multimedia compression, all operations are performed according to agreed standards, which allows the raw content to be decoded from the compressed bitstream. The comparison between encryption and compression is listed in Table 1.

Based on this observation, we argue that encryption can be achieved by controlling certain operations in the compression system using a secret key. As a result, a correct key is required to decode the bitstream and recover the original multimedia content, just as one cannot obtain the original plaintext from the ciphertext without knowing the encryption key. If it is properly designed, such a scheme would demand a low computational cost since operations such as motion estimation, DCT, quantization have already taken care of the heavy work of redundancy removal from the input data. The focus of the remaining design is to manipulate the output bitstreams so that the resultant encryption scheme achieves high security. We stress here that such an encryption scheme should meet at least the following criteria.

(1) High security
The scheme should provide resistance against various types of attacks, including the ciphertext-only attack and the known/chosen plaintext attack.

(2) Low encryption cost
The encryption cost should not exceed an acceptably small portion of the total computation cost of compression (motion estimation, DCT transform, quantization, etc.). In most practical applications, 5% could be a proper threshold.

(3) No harm to the compression ratio
The ultimate goal of multimedia compression is to reduce the bitstream length to the minimum possible extent. Any multimedia encryption scheme cannot violate this fundamental goal. Achieving high security at the expense of sacrificing the compression ratio is not desired. Again, we may consider a proper threshold depending on the application context. For example, the increase of the final bitstream size due to encryption should not be higher than 5% of the original coded bitstream.

(4) Compatible to standard compression.
It is desirable that an encryption scheme can go back to the standard compression by a simple configuration using a trivial key (say, a key with the value of zero). This provides users flexibility since they can decide whether or not to perform encryption according to the security concern of specific applications.

In what follows, we propose two novel techniques to maneuver the compression system, and the combination of the two can form an efficient and secure multimedia encryption solution. The first method is called *randomized entropy coding* (REC). REC uses multiple coding parameters/settings and dynamically chooses one to encode each successive symbol according to a random sequence. In contrast, standard entropy coding has only one parameters/settings in the entire encoding. The REC method is an extension of previous work by Wu and Kuo [10, 15]. The second technique is called *rotation in partitioned bitstream* (RPB). It is cascaded after the REC module to further scramble the bitstream encoded by REC. As the name suggests, RPB first partitions bitstream into blocks and then performs a random cyclic rotation in each block. The joint REC/RPB encryption paradigm is illustrated in Figure 1.

The box "compression before entropy coding" represents all operations before entropy coding, including motion compensation, DCT transform, quantization, and so forth. Its

Table 1: Comparison of encryption and compression.

	Encryption	Compression
Input redundancy	High	High
Output redundancy	Low	Low
Output size	= input size	< input size
Redundancy removal operations	AND, OR, XOR, shift vector, matrix multiplication permutation, substitution	Motion estimation DCT, quantization entropy coding
Decryption/Decoding	*secret key required*	*no key required*

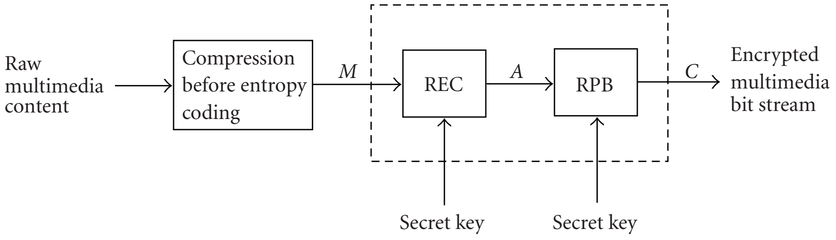

Raw multimedia content
Compression before entropy coding
M
REC
A
RPB
C
Encrypted multimedia bit stream
Secret key
Secret key

Figure 1: The joint REC/RPB encryption scheme.

output M are symbols in compressed domain such as DCT coefficients and motion vectors. The REC module encrypts M to an internal ciphertext A, which is further processed by the RPB module to produce the final encrypted bitstream C. REC and RPB modules are enclosed by dotted line to emphasize the fact that in practice they are implemented as a whole inside the entropy coder. The dotted line box conceptually behaves like a black box and the internal ciphertext A cannot be observed from outside. We will analyze later in Section 7 how this affects the model's security to resist cryptographic attacks.

Throughout the rest of the paper the following notations are used.

$x \leftarrow y$: x assigns the result of evaluating y
$a[i]$: the ith leftmost bit of binary string a
$a \| b$: the concatenation of binary string a and b
$a \ll r$: the r-bit left shift operation on binary string a
$a \gg r$: the r-bit right shift operation on binary string a
$\lceil a \rceil$: the smallest integer larger than $a > 0$
$\{0, 1\}^n$: the space of all n-bit binary strings
$h(\cdot)$: cryptographic one-way hash function
PRBG: cryptographic pseudorandom bit generator.

4. RANDOMIZED ENTROPY CODING (REC)

A question following the discussion in the last section is what are the ideal operations/steps that can be controlled using a secret key so as to achieve encryption? Wu and Kuo [10, 15] are the first to explore in this direction and they proposed to implement encryption in entropy coding. In standard entropy coding, only one statistical model (though it may adapt to varying input statistics) is used throughout the whole encoding process. It is their novel idea to use multiple statistical models to encode each individual symbol while the order of those multiple models are kept secret as the key. Since choosing a random model usually demands only a negligible computation cost, encryption can be done very quickly. They proposed two encryption schemes called the multiple Huffman table (the MHT coder) for the Huffman coder and the multiple state indices (the MSI coder) for the QM coder.

In this section, we extend this multiple statistical model coding method and develop the concept of *randomized entropy coding* (REC). It is readily observed that other than statistical model, there exist other adjustable parameters/settings in the entropy coding. Changing these parameters during entropy coding will lead to different bitstream output. One example is the use of different quantization table to generate bitstreams with variable rates in a VBR (variable bit rate) coding scheme. We can make further distinction between two type of adjustable parameters.

(i) This first type of parameters adjust their values according to statistical property of input. Their values change dynamically to better accommodate the change of input statistics and are closely related to the coding efficiency of the entropy coder. For instance, the probability estimation in an adaptive QM coder is determined by an internal state machine and changes according to the state and current input.

(ii) The second type of parameters has nothing to do with coding efficiency. Instead, they are chosen as a general setting of the entropy coder and the particular choice is just a matter of preference or convention. The Huffman tree in the Huffman coder is an example of this type of parameters. We can use different binary codes to implement the same Huffman tree.

Because the second type of parameters does not affect the coding efficiency of entropy coder, they are obviously ideal choices in REC encryption method. We make a formal definition below.

Definition 1 (equivalent coding paramter). An equivalent coding parameter (ECP) is a parameter in the entropy coder that meets the following conditions:

(1) using different (often adjustable) values of this parameter will lead to different bitstream output;

(2) changing values of this parameter dynamically during coding does not affect the coding efficiency.

We use the word "equivalent" to emphasize the fact that an ECP can take different values freely during entropy coding and the choice does not have an impact on the coding efficiency. By default, an entropy coder uses a fixed value of ECP to encode all inputs throughout the entire compression process. In our proposed REC approach, a particular ECP value is selected according to a random sequence to encode each individual input. This random sequence apparently becomes the encryption key since it is needed in order to correctly decrypt the bitstream. This sequence is termed the *key hopping sequence* (KHS) in that the way REC works is similar to a frequency hopping communication system. The entropy coder alternates among different ECP values just as the communication channel hops among different frequencies according to a random sequence. Apparently, the property of KHS is of utmost significance to the security of the REC encryption approach. One has to be cautious in designing a good KHS to achieve a high level of security.

Let us examine the desired properties of a KHS. Note that REC encryption can be viewed as a successive series of random tests, each step being choosing a random ECP value according to the KHS. Thus, the first requirement is that the KHS be indistinguishable from a truly random sequence statistically. An attack should not be able to differentiate it from a truly random sequence based on statistical properties such as the mean, the variance, and the distribution of run length, and so forth. Second, successive bits of a KHS should be statistically independent. This is because it is always prudent to assume that an attacker is able to obtain part of the KHS being used. The statistical independence between successive bits prohibits attacker from gaining any useful information about other parts of KHS. These two requirements can be expressed as follows.

(1) Given a KHS and a truly random sequence of the same length, no polynomial-time algorithm can distinguish them apart with probability significantly greater than 1/2.

(2) Given a sequence of k bits of a KHS, no polynomial-time algorithm exists that can predict the $(k + 1)$th bit with a probability significantly greater than 1/2.

In cryptography, the above two conditions are recognized as the *polynomial-time statistical test* and the *next-bit test*, respectively [24]. It is also well known that a pseudorandom bit sequence meets these two conditions and such a sequence can be generated by a *pseudorandom bit generator* (PRBG). The input of a PRBG is a relatively short binary sequence generally called the *seed*, which drives the PRBG to output a very long pseudorandom bit sequence.

Next, we present two encryption schemes based on the REC model. They are associated with the well-known Huffman coder and arithmetic coder, respectively.

4.1. Randomized Huffman table (RHT) scheme

Huffman coding is the most widely used entropy coder in image/video compression system. The Huffman tree is a good ECP since the same tree can be represented by different binary codes. The RHT scheme is actually very similar to that in [10] and it was presented here as an example under the REC model. In the RHT encryption, a number of different Huffman codes are constructed that correspond to the same Huffman tree and published. This can be easily done using a technique called the Huffman tree mutation process [10]. Then, a particular Huffman codes is chosen to encode each input according to the KHS. The detailed algorithm is described below.

RHT encryption scheme:

(1) Generate $M = 2^m$ different Huffman coding tables, numbered from 0 to $M - 1$. These tables can be made public.

(2) Select a cryptographically secure PRBG as the KHS generator. Generate a random seed s, which is the key of RHT encryption. $z \leftarrow$ first output of KHS generator.

(3) Break z into m-bit blocks. Write $z = t_1\|t_2\|\cdots\|t_k\|rem$ with each t_i representing a number from 0 to $M - 1$ and rem the remaining bits.

(4) for $i = 1$ to k
use Huffman table t_i to encode one symbol.

(5) After encoding k symbols in Step (4), update KHS: $z \leftarrow$ new output of KHS generator. Go to Step (3).

The legitimate receiver knows the key (random seed s). He is thus able to reproduce the KHS used in encryption and in turn correctly decode the bitstream. We give an example of RHT encryption scheme below to illustrate several interesting properties.

We assume a small alphabet of the source input consisting of seven symbols, denoted by A, B, C, D, E, F, G. Two different Huffman codes, as shown in Figure 2, are constructed to encode these 7 symbols. Note that the topologies of two Huffman trees are the same so the code length of each symbol is identical, although the code values are different. A sample input plaintext

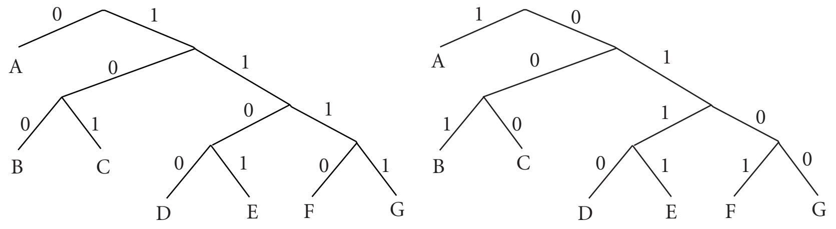
$$P = \text{ACDABEFG}, \quad (2)$$

is encrypted using two KHS sequences

$$k_1 = 00000000, \qquad k_2 = 10011010, \quad (3)$$

where "0" indicates that Huffman code #0 is used to encode the plaintext symbol while "1" indicates the use of Huffman code #1. Note that the all-0 key k_1 corresponds to the default Huffman coding where code #0 is used to encode all plaintext inputs. The key value and the corresponding ciphertext are shown in Table 2 with different ciphertext bits highlighted by the blue color. It is clear that the difference depends on the particular key value chosen.

Assume that plaintext P is encrypted using key k_2 with the ciphertext as shown in the 2nd row of Table 2. Next, we



Huffman code number 0

(a)

Huffman code number 1

(b)

Figure 2: Two Huffman trees with the same topology.

Table 2: RHT encryption using two different keys.

Plaintext	KHS	Ciphertext
ACDABEFG	00000000	0101110001001101111101111
	10011010	1101110010011101010111111

Table 3: RHT decryption using three different keys.

Ciphertext	KHS	Plaintext
1101110010011101010111111	10011010	ACDABEFG
	00000000	EDBFCAG
	10111010	ACAABAEA

study the effect of the RHT decryption with 3 keys as shown in Table 3. The first KHS is the correct one so that it recovers plaintext P successfully. The second KHS is the all-0 sequence which emulates the situation where the receiver decodes the RHT-encrypted ciphertext using the standard Huffman decoding procedure. The decoding result is totally different from the correct plaintext P. Furthermore, it is important to note that even if only 1 bit in the KHS is wrong, the decoding result starting from that position will be totally wrong. This error propagation effect is demonstrated by the third KHS. The third KHS is different from the correct one only at the 3rd bit. The first 2 plaintext symbols are decrypted correctly. However from the 3rd plaintext symbol on, the decryption result totally deviates from the correct plaintext P. Since Huffman code is a unique decodable code, decoding can always continue with any KHS sequence. This decoding error will not be detected until the wrong results are further converted to raw multimedia content and found meaningless.

Finally, it is worthwhile to point out that the construction of different Huffman tables plays an important role in security. A design guideline is to ensure that any symbol has an association with at least two different bit sequences in the union of all possible Huffman tables. Otherwise, an attacker would be able to produce a particular output in a chosen plaintext attack. For instance, if we do not swap the 0-1 labeling on the root in Figure 2, symbol A will correspond to code “0” in both code #0 and code #1. Then, an attacker can easily generate an output $0000 \cdots 0$ by inputting sequence $AAAA \cdots A$. Such a particular pattern could be used to mount a powerful attack to the following RPB module. As discussed later, security analysis in Section 7.2 assumes that the output of the REC module can be viewed as a random bit sequence. This design guideline must be strictly enforced for the assumption to be valid.

4.2. Randomized arithmetic coding convention interleaving (RACCI) scheme

The binary arithmetic coder is another popular entropy coding method widely used in multimedia compress system. Simply speaking, arithmetic coding is a process of repeatedly dividing an interval, and any point in the current interval represents the bitstream. There have been previous research on using adaptive arithmetic coding as a means of encryption. But those schemes are not satisfactory in terms of both security and complexity. (Please refer to [25–28] for discussion of those schemes and security analysis). Based on the REC approach, we propose an encryption scheme called random arithmetic coding convention interleaving (RACCI) encryption. This scheme is first developed in one of the author’s early work [17] and we show here that it can fit into the REC model. As the name suggests, the ECP we have chosen for this scheme is the coding convention in arithmetic coding.

In binary arithmetic coding, there are two possible symbol orderings (i.e., the LPS subinterval above the MPS subinterval, or the MPS subinterval above the LPS subinterval) and two possible code stream conventions (i.e., points to the bottom or the top of an interval), which leads to a total of four possible coding conventions. In the following, we use QM coder to illustrate the technical details of RACCI encryption. QM coder represents a well-known binary arithmetic coder that uses techniques such as multiplication approximation and renormalization of the probability interval to optimize performance. Here, C denotes the bitstream and A is the updating inteval, Q_e is the probability of the least probable symbol. Figure 3 illustrates these 4 coding conventions.

Convention (a):

if MPS: C unchanged, $A = A - Q_e$,
renormalize if needed
if LPS: $C = C + A - Q_e$, $A = Q_e$,
renormalize.

Convention (b):

```
if MPS: $C = C + Q_e$, $A = A - Q_e$,
    renormalize if needed
    if LPS: $C$ unchanged, $A = Q_e$,
    renormalize.
```

Convention (c):

```
if MPS: $C = C - Q_e$, $A = A - Qe$,
    renormalize if needed
    if LPS: $C$ unchanged, $A = Q_e$,
    renormalize.
```

Convention (d):

```
if MPS: $C$ unchanged, $A = A - Q_e$,
    renormalize if needed
    if LPS: $C = C - A + Q_e$, $A = Q_e$,
    renormalize.
```

Only conventions (a) and (b) are used in our proposed scheme. Because although conventions (a) and (c) look very different, the difference between the two bitstreams is always equal to the remaining probability interval A, as can be seen by careful inspection of (4) and (6). There is a similar relationship between the code streams of conventions (b) and (d). The proposed RACCI encryption scheme is described below.

RACCI encryption scheme:

(1) Select a cryptographically secure PRBG as the KHS generator. Generate a random seed s, which is the key of RACCI encryption.

(2) $z \leftarrow$ output of KHS generator.

(3) For the ith input
    if $z[i] = 0$

        use convention (a) to encode the input

    if $z[i] = 1$

        use convention (b) to encode the input.

(4) Repeat Steps (3) until all inputs are coded.

The legitimate receiver knows the key (random seed s). He is thus able to reproduce the KHS used in encryption and in turn correctly decode the bitstream.

5. ROTATION IN PARTITIONED BITSTREAM (RPB)

The idea of the RPB encryption first appeared in [22], which used two operations in cascade to encrypt a compressed bitstream. The 0-1 bitstream is first partitioned into blocks of random sizes and then a circular random rotation is performed within each block. We revisit the RPB encryption and provide more analytical results in this section. In particular, an interesting concept called the equivalent key, which is important in defending the known/chosen plaintext attack, is developed and its properties are investigated.

Many operations can be used to alter the bit order in a block. A permutation on all bits shuffles the bit order most thoroughly but requires a lot of computation. To reduce the complexity and facilitate the bitstream processing, we restrict the bit manipulation to a simple *left rotation* here. For a block of n bits $A = (a_1 a_2 \cdots a_n)$, an r-bit left rotation transforms A into $(a_{r+1} a_{r+2} \cdots a_n a_1 a_2 \cdots a_r)$ by rotating the first r bits to the end of A. The main reason to use this simple operation is that it can be easily merged into the algorithm that prepares the bitstream for the final output, thus adding a very small computation overhead. Furthermore, although left rotation is a simple operation, our analysis in Section 7 shows that, if being combined with random-sized block partitioning, it does provide high security. Mathematically, the above concept can be formalized as follows.

Definition 2. Let $A = (a_1 a_2 \cdots a_N)$ be a bitstream of length N. The (p, r) rotation in partitioned blocks of A, denoted $\text{RPB}(A, p, r)$ with $p = (p_1 p_2 \cdots p_m)$ and $r = (r_1 r_2 \cdots r_m)$, is obtained by the following 2 steps.

(1) Partition A into m blocks A_i with length p_i, $i = 1, 2, \ldots, m$, $\sum_{i=0}^{m} p_i = N$.

(2) Perform an r_i-bit left rotation on each block A_i, $i = 1, 2, \ldots, m$.

An example is given in Figure 4 to illustrate the RPB operation applied to a stream of 10 bits $A = (a_0, a_1 \cdots a_9)$. The partition sequence is $p = (3, 5, 2)$ and the rotation sequence is $r = (2, 3, 1)$. The bitstream after performing $\text{RPB}(A, p, r)$ is denoted by C.

In the proposed RPB encryption scheme, a plaintext bitstream A is enciphered into a ciphertext $\text{RPB}(A, p, r)$ with the partition sequence p and rotation sequence r. To achieve the best possible random scrambling, it is important that sequences p and r are highly random without much statistical regularities. For this reason, components p_i and r_i are obtained from a pseudorandom bit sequence, which is generated by a PRBG using a secret seed.

The RPB algorithm has another performance advantage. In real-world data compression system, coded bits output from the entropy coder are first sequentially queued into a buffer. Only after enough number of bits has accumulated in the buffer, the buffer will be written to the final compressed data file so as to avoid frequent memory access. This allows the RPB operation to be conveniently implemented by simply regulating the order in which bits are queued into the buffer. For a single p-bit block A, an r-bit left rotation is equivalent to a "hold-and-write" operation as specified in the following steps:

(1) hold the first r bits of A;

(2) write the remaining $p - r$ bits to the buffer;

(3) write the r bits in Step (1) to the buffer.

The above "hold-and-write" procedure enables to perform the RPB encryption instantaneously as coded bits are continuously generated from entropy coder. Furthermore, the size of the output buffer is finite in the real world implementation. It is assumed to be bounded by B bits. To accommodate

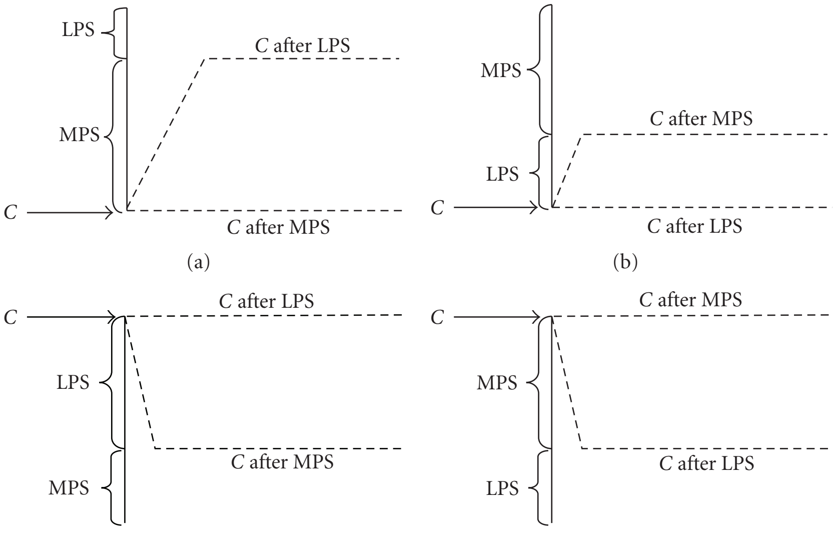


Figure 3: Four possible coding conventions of arithmetic coding.

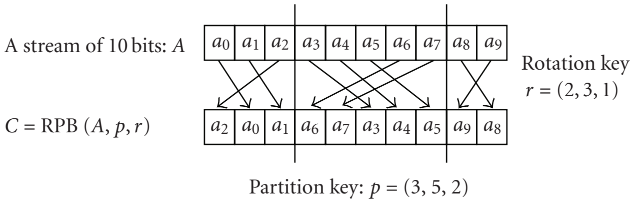

A stream of 10 bits: A
a0 a1 a2 a3 a4 a5 a6 a7 a8 a9
Rotation key
r = (2, 3, 1)
C = RPB (A, p, r)
a2 a0 a1 a6 a7 a3 a4 a5 a9 a8
Partition key: p = (3, 5, 2)

Figure 4: An example of rotation in partitioned bitstream.

the "hold-and-write" operation described above, it is clear that the block partition size p_i cannot exceed the output buffer size; namely, $p_i < B$.

The proposed RPB encryption algorithm is outlined as follows.

Rotation in partitioned bitstream (RPB) scheme

(1) Select a secure PRBG algorithm and generate a random number s as the seed (which is also the encryption key). The output keystream z is grouped into B-bit blocks to produce a random number in the range $0\sim 2^B - 1$.

(2) Obtain two random numbers p and r' from z, and scale r' into the range $0\sim p$ by computing $r = (p \times r') \gg B$.

(3) Hold the first r bits of the output bit stream from the entropy coder.

(4) Write next $p - r$ bits of the output bit stream to the buffer. Then, write the r bits in Step (3) to the buffer.

(5) When the buffer is full, write the buffer content to the final bit stream file.

(6) Repeat Steps (2)$\sim$(5) until no more bits are output from the entropy coder.

The secret seed s is the encryption key and $C = \text{RPB}(A, p, r)$ is the ciphertext bit stream. On the receiving side, sequence z with its component partition sequence p and rotation sequence r can be generated using the same encryption key. It is easy to check that operation $\text{RPB}(C, p, p - r)$ recovers the plaintext A from the ciphertext C.

Next we investigate several important mathematical properties of the RPB operation. As will be shown later in Section 7, these properties form the basis of analyzing security under various types of attacks.

5.1. Key space analysis

We first study the key space size of RPB encryption. For a given N-bit ciphertext $C = \text{RPB}(A, p, r)$, the key space of the RPB scheme is the total number of different ways to decrypt C using all possible partition sequence p and rotation sequence r. As mentioned before, if the ciphertext is $C = \text{RPB}(A, p, r)$, then the plaintext is $A = \text{RPB}(C, p, p - r)$. Thus, the key space is equivalent to the total number of different ways to encrypt A using all possible p and r. We have the following definition.

Definition 3. Let $A = (a_1 a_2 \cdots a_N)$ be a bitstream of length N. Two RPBs of A, $\text{RPB}(A, p_1, r_1)$ and $\text{RPB}(A, p_2, r_2)$, are said to be different if they achieve a different order of a_i's in the resulting stream C. The total number of different RPBs is denoted by $R(N)$.

The key space of a complete permutation of $A = (a_1 a_2 \cdots a_N)$ is $N!$. Clearly, $R(n) < N!$ because a lot of these permutations cannot be achieved by applying RPB operation due to two reasons. First, the block rotation in RPB operation prohibits some particular permutations to be produced. For example, in a simple case $A = (a_1 a_2 a_3 a_4)$, the permutation $(a_4 a_3 a_2 a_1)$ cannot be a result of any RPB operation.

Actually $R(4) = 12$ while the number of complete permutation is $4! = 24$. Second, the upper bound of the partitioned block size reduces the number of different RPBs. Because we require $p_i < B$, it is impossible that an RPB starts with a_i for $i > B + 1$.

While an exact expression of $R(N)$ may be difficult to obtain, we derive a recursive relationship of $R(N)$ and establish a lower bound for $R(N)$ as given in the following lemma.

Lemma 1. *Let $A = (a_1 a_2 \cdots a_N)$ be a bitstream of length N and B the maximal length of partitioned blocks A_i. Then, the total number of different RPBs of A, denoted by $R(N)$, satisfies the following two equations:*

$$R(N) = 2R(N-1) + \sum_{k=N-B}^{N-3} R(k), \tag{4}$$

$$R(N) > 2^N, \quad for\, N \geq 6. \tag{5}$$

The basic idea is to divide all possible RPBs into B categories according to the first bit being a_1, a_2 up to a_B. Then, the number in each category is counted and summed up to get (4). From this recursive equation, the lower bound given in (5) is straightforward since $R(N) > 2R(N-1)$. A detailed proof is provided in Appendix A.1.

It is important to observe that the size of $R(N)$ grows exponentially with the length of the plaintext/ciphertext. For a large value of N, it becomes impractical to exhaust all possible RPBs for a given ciphertext.

5.2. *Equivalent key analysis*

We studied $R(N)$, the total number of possible RPBs of a stream of N bits, and provided a lower bound for $R(N)$ in the last subsection. In this subsection, we analyze another interesting property of RPB, called the equivalent key, and show how it can help defend known/chosen plaintext attack.

In Definition 3, two RPBs are different if they lead to a different order of a_i's in the resulting stream C, where all a_i's are treated as distinct symbols. If two RPBs yield different ciphertext C, then they must be different. However, the converse is not always true, that is, two different RPBs may transform A to the same ciphertext C. This is due to the fact that, when the plaintext A is a binary bitstream, each a_i is either 0 or 1. Therefore, it is possible that two different RPBs give the same ciphertext, although the underlying order of a_i's is different. This effect can be explained by the following example.

Example 1. 8-bit plaintext: $A = (a_1 a_2 \cdots a_8)$
key 1: $p_1 = (1, 7)$, $r_1 = (0, 1)$
key 2: $p_2 = (3, 4, 1)$, $r_2 = (2, 1, 0)$.

For the above two keys, it is readily checked that $\text{RPB}(A, p_1, r_1) = (a_1 a_3 a_4 a_5 a_6 a_7 a_8 a_2)$ and $\text{RPB}(A, p_2, r_2) = (a_3 a_1 a_2 a_5 a_6 a_7 a_4 a_8)$. They are apparently different RPBs by Definition 3. However, for a particular plaintext $A = (01011101)$, we have $\text{RPB}(A, p_1, r_1) = \text{RPB}(A, p_2, r_2) = (00111011)$. That is, both keys encipher A to the same ciphertext $C = (00111011)$. These keys are called *equivalent keys*. Mathematically, the equivalent key is defined as follows.

Definition 4 (equivalent keys). For a given plaintext bitstream A, two keys (p_1, r_1) and (p_2, r_2) are called equivalent keys if

(1) $\text{RPB}(A, p_1, r_1)$ and $\text{RPB}(A, p_2, r_2)$ are different RPB per Definition 3,
(2) they transform A to the same output $C = \text{RPB}(A, p_1, r_1) = \text{RPB}(A, p_2, r_2)$.

We stress that the concept of equivalent keys is associated with a particular ciphertext (assuming a fixed plaintext). Two equivalent keys for one ciphertext may not be equivalent keys for another ciphertext. Discussion on equivalent keys is not meaningful without the context of one particular ciphertext. Given a plaintext/ciphertext pair, it is natural to consider two important questions regarding equivalent keys. First, does there exist equivalent keys? Second, if there is any, then what is the exact amount of equivalent keys for the given pair?

The answer to the first question is most likely positive since one is allowed to arbitrarily partition the bit stream provided that block size $< B$ and rotate freely in each block. From the above 8-bit plaintext example, it seems not so hard to obtain two equivalent keys by observing the bitstream pattern and do several trials. The second problem, that is, to compute the accurate number of equivalent keys, is however not an easy one. Since equivalent keys are ciphertext dependent, there seems no quick formula to compute the number of equivalent keys for a given plaintext/ciphertext pair. Nonetheless, if we take into account all possible ciphertexts C for a plaintext A, we have the following conclusion regarding equivalent keys.

Lemma 2. *Let $A = (a_1 a_2 \cdots a_N)$ be a bitstream of length N containing Z 0's and let* $\text{Equiv}(A, C)$ *denote the number of equivalent keys for the plaintext/ciphertext pair (A, C). Then, there exists a ciphertext C' such that*

$$\text{Equiv}(A, C') > \left\lceil 2^N \Big/ \binom{N}{Z} \right\rceil. \tag{6}$$

In a statistically average sense, a random plaintext A contains half 0's ($Z = N/2$). When the plaintext length N is large enough, we have

$$\text{Equiv}(A, C') > \sqrt{\pi N/2}. \tag{7}$$

The above lemma establishes the existence of equivalent keys. Refer to Appendix A.2 for a complete proof. The quantity $\sqrt{\pi N/2}$ is however a conservative estimate of number of equivalent keys. Further analysis of the average number of equivalent keys will be given in Section 7.3.

6. COMPUTATIONAL COST ANALYSIS

The computational cost of REC encryption consists primarily of two parts: the KHS generation cost, and the cost to have entropy coder dynamically select an ECP value. Usually the first part is the major computational overhead because the length of KHS required to encrypt all inputs is proportional to the length of the plaintext M. As for the second part, if the entropy coder is implemented in software, this can be done by adding a variable index (according to KHS) to the base address of ECP value. It takes no more than 2 to 3 instructions to accomplish this task. If the entropy coder is implemented by hardware, then this cost translates to several kilobytes of memory to store multiple ECP values in an array plus a couple of multiplexer and control logic to index into the array. In general, this part of the cost is much lower as compared to the KHS generation cost.

The RPB encryption scheme is in essence a bit reordering algorithm in variable-length blocks of the plaintext bitstream. In contrast to cryptographic ciphers, there are no multiple rounds of complicated bit manipulation operations invoked by the RPB scheme. Encryption is achieved by the simple "hold-and-write" operation described in the last section.

In practice, it is quite easy to implement the "hold-and-write" operation in parallel with the algorithm that forms the bit stream. The only addition needed is a small delay buffer (less than B bits). First, hold r bits output from the entropy coder in the delay buffer. Then write next $p - r$ bits from the entropy coder into the output buffer. Finally, write the r bits in the delay buffer into the output buffer. Since this can be easily done either by software or hardware, the overhead of implementing the RPB scheme in a multimedia compression system is almost negligible. Actually, the primary encryption cost is the generation of pseudorandom sequences to yield the partition sequence and the rotation sequence.

7. SECURITY ANALYSIS

In this section we discuss the security strength of the joint REC/RPB encryption paradigm under three most common cryptographic attack types: ciphertext-only attack, known plaintext attack, and chosen plaintext attack. As shown in Figure 1, in our system the ciphertext is C, the output of RPB module. The plaintext could be regarded M, the direct input to the REC module, because M can be converted to/from the raw content using standard decoder/encoder. As to the key, we consider the KHS used in REC and the partition and rotation key sequence used in RPB, but not the random seed of the PRBG generator, as the key of interest. Recovering these key sequences (or a large part of them) is deemed a successful attack because these sequences allow directly decrypting C to M, which could be decoded to raw content using standard decoder.

We stress that in our system, the output of REC module A (also input to RPB module) in Figure 1 is not available to the adversary for study. Although conceptually REC and RPB are two modules, in practice they are easily implemented together in the entropy coder as a whole. Therefore, A as an "internal" ciphertext is usually not accessible to outside entity. In other words, the adversary can arbitrarily manipulate the input M and observe the output C. But he does not have the capability to obtain the value of A nor insert an arbitrary A of his choice in between the REC and RPB encryption.

7.1. *Ciphertext-only attack*

In this attack the adversary is given only the ciphertext C and tries to deduce the key or plaintext M. Adversary can pick a random partition/rotation key sequence to decrypt C to a possible A, then pick another random KHS, decrypt that A to M, and finally decode M to see whether the raw content is meaningful. The computation involved is quite heavy. Since adversary has no idea what the value of actual A is, he has to examine all possible A in the first step and all possible M in the second step. As shown by Lemma 1, the key space of the first step already amounts to $R(N) > 2^N$, not to mention checking all possible M for each A in second step. Given this exponential key space, the bitstream in real applications is usually long enough to thwart any ciphertext-only attack. For instance, in the state-of-the-art video compression standard such as H.264, it would cost around 1~2 kilobits to encode a CIF-size (352 × 288) video frame.

An adversary could also exhaust all possible values of PRBG seed that generates the KHS and partition/rotation sequence. The search space for an r-bit number is 2^r. Considering the current state-of-the-art of computing, using seed longer than 80 bits in our encryption provides adequate safety margin under ciphertext-only attack.

7.2. *Known plaintext and chosen plaintext attack*

In the known plaintext attack, several M/C pairs are available for study. With the knowledge of the plaintext M, the adversary can launch a classic "meet-in-the-middle" attack on the internal ciphertext A. Starting from the plaintext side, the adversary picks random KHS and REC encrypts M to A_1. On the ciphertext side, the adversary chooses random partition/rotation keys and RPB decrypts C to A_2. The adversary accumulates two datasets A_1 and A_2 until a collision $A_1 = A = A_2$ is found. Let us study the complexity of this attack to find the internal ciphertext A.

Due to the pseudorandom KHS and entropy coding property, the output of REC module A can be generally considered a random N-bit sequence. The same conclusion applies to C given the randomness of the partition and rotation key. This can be justified by the experimental study in Section 8.3 that entropies of A and C are very close to 1 bit/symbol, the entropy of a truly random binary sequence. Based on this and the random selection of KHS and partition/rotation keys in the above attack, A_1 and A_2 could be regarded as a random sample from the space of all N-bit sequence as well. This is a classic birthday attack and the computational complexity (i.e., expected number of trials before a collision are met) is $(2^N)^{1/2} = 2^{N/2}$. Similar to the discussion in ciphertext-only attack, the adversary would rather resort to an exhaustive search on the seed given the large size of N. Suppose the seed length for KHS generator is r_1 and r_2 for partition and rotation key generator. Then the complexity is clearly $2^{r_1} + 2^{r_2}$.

In the chosen plaintext attack, the adversary has the additional freedom to select any plaintext M of his/her choice and study the corresponding ciphertext C. Note that RPB is basically a simple bit-reordering scheme. If we allow the input to the RPB module, A, to be arbitrarily manipulated, then the partition and rotation keys could be determined by applying inputs of a particular pattern. One such attack algorithm was suggested by Chia-Mu Yu at the Institute of Information Science, Academia Sinica, Taiwan, to the authors through personal communication. This algorithm requires $O(N)$ chosen plaintexts and $O(N)$ time complexity. Thus, the RPB encryption as a stand-alone module cannot withstand the chosen plaintext attack. This is however not the case in the joint REC/RPB model as emphasized in the beginning of Section 7. Although the adversary can freely choose M, he/she still does not have the capability to manipulate the value of actual A directly due to the random KHS used in the REC encryption and the fact that the value of A is not accessible. Therefore, a chosen plaintext attack does not bring in much advantage as compared to the known plaintext attack.

In summary, inaccessibility of the internal ciphertext due to the joint REC/RPB model as a black box inside the entropy coder has played a crucial role in the strength of the proposed encryption scheme to resist attacks. This is an inherent advantage of the joint REC/RPB encryption paradigm. As a result the attack complexity is exponential to the length of the seed. Therefore, using a sufficiently long seed (> 80 bits) ensures the security of the proposed encryption scheme.

7.3. More on equivalent key

Having made the above discussion, let us assume a scenario where the adversary is able to observe the value of A for a given ciphertext C by whatever means. We study the security of our joint REC/RPB encryption under this attack.

As pointed out earlier in Section 5.2, there exist multiple equivalent keys that encipher an input A to the same ciphertext C. Thus, an adversary cannot differentiate these equivalent keys given only a few A/C pairs. Instead lots of pairs are required in order to uniquely determine the correct key. Of course, the larger the number of equivalent keys for a general ciphertext, the more pairs are needed and hence the greater the complexity to determine the correct key. Security under this attack thus directly relies on the number and characteristics of equivalent keys for a general plaintext/ciphertext pair.

It is shown in Lemma 2 that the number of equivalent keys Equiv (A, C) is larger than $\sqrt{\pi N/2}$ for at least one ciphertext C'. This is however a conservative worst-case estimate for two reasons. First, the actual size of $C(N)$ is strictly less than $\binom{N}{N/2}$, yet $\binom{N}{N/2}$ was used in the derivation. Second, it is implicitly assumed (by the pigeon hole principle) that equivalent keys of different ciphertexts do not overlap. In fact, we can show by plausible reasoning that the number of equivalent keys far exceeds $\sqrt{\pi N/2}$. It also grows exponentially with respect to the ciphertext length N.

In analysis below, we denote the statistical average number of equivalent keys for a general N-bit plaintext by $E(N)$. We have the following property regarding $E(N)$.

Lemma 3. *$E(N)\sim c^N$ for sufficiently large N, where $c > 1$ is a constant.*

The key to prove this conclusion is the observation that if k_1 is any one of the $E(N)$ equivalent keys for a general plaintext A_1 and k_2 is any one of the $E(N)$ equivalent keys for another plaintext A_2, then a concatenation key $k = k_1 \parallel k_2$ is an equivalent key for the plaintext concatenation $A_1 \parallel A_2$. A function $E(N)$ satisfying this property must be of the form c^N. Appendix A.3 gives a detailed proof.

We emphasize that c^N is not an accurate formula of $E(N)$ but it depicts the asymptotic behavior of $E(N)$ with respect to N. For a sufficiently large value of N (a long enough plaintext), the average number of equivalent keys for a general plaintext quickly becomes intractable since it is exponential with the plaintext length. This exponential growth rate of the number of equivalent keys, $E(N)$, plays a key role in RPB encryption's ability to withstand this attack.

Due to the large amount of equivalent keys, an adversary cannot determine the correct key given only few plaintext/ciphertext pairs. However, it is interesting to ask, when a sufficient number of plaintext/ciphertext pairs are available for analysis, whether it is possible to exclude wrong equivalent keys and determine the correct key uniquely. Thus, we study this problem and estimate the number of plaintext/ciphertext pairs needed to launch a known plaintext attack. The total computational cost of this attack is analyzed.

Given a sufficient number of plaintext/ciphertext pairs, an adversary can proceed as follows. At first, he/she can select a given pair and calculate $E(N)$ equivalent keys for this pair using a certain algorithm. Then, he can check all these $E(N)$ keys against each available pair (A_i, C_i). If $\mathrm{RPB}(A_i, k) = C_i$, then k is counted as one possible key. Otherwise, k must be a wrong equivalent key, which can be discarded. As more and more pairs are examined, the number of possible keys decreases. This process is repeated until only one key is left, which must be the correct key. For a general N-bit plaintext, let $P(N)$ denote the expected number of plaintext/ciphertext pairs needed to uniquely determine the correct key. The following lemma offers an estimate for $P(N)$.

Lemma 4. *With $R(N)$, $E(N)$, and $P(N)$ defined in the above description, one has the following relationship:*

$$P(N) = \frac{R(N)}{R(N) - E(N)} \ln R(N). \tag{8}$$

Then, a rough estimate of $P(N)$ is given by

$$P(N) \approx N \ln 2. \tag{9}$$

To solve this problem, we may examine it from another angle. That is, X containing only one key is equivalent to saying that $\overline{X}$, the complementary set of X, contains $R(N) - 1$ keys. By treating $\overline{X}$, we can convert this problem to a variant of the classical coupon collector problem. Please refer to Appendix A.4 for a complete proof.

Although the number of plaintext/ciphertext pairs needed to uniquely determine the correct key is linear with N, the total complexity to mount such an attack is still formidable due to the exponential growth rate of equivalent key number $E(N)$ with plaintext length N. Remember that the adversary needs a certain algorithm to first find out all

$E(N)$ equivalent keys and then has to check for each plaintext/ciphertext pair in order to eliminate wrong keys. It was assumed before that in this attack, an adversary is only allowed to observe A but prohibited from directly manipulating A. In other words, it is difficult for an adversary to produce certain A with an arbitrarily desired characteristics for advanced attacks. Thus, we can claim that, for sufficiently large N, which is true for multimedia data, the cascaded REC/RPB scheme provides strong resistance against the known/chosen plaintext attack even when the adversary is allowed the extra capability to observe the internal ciphertext A.

7.4. *RPB's impact on statistical randomness of A and C*

In the above sections, we study the security of the joint REC/RPB encryption scheme under several attack modes from a cryptographic viewpoint. In this section, the security of our scheme is examined from another angle. We study RPB's impact on the statistical randomness of input A and output C. It is apparent that our encryption should not weaken the statistical randomness of its input A. In other words, the output C should be at least as random as the input stream A under some statistical measure. For binary sequences, a commonly used measure of their randomness is the entropy. Thus, we would like to study the relationship between the entropies of A and C. This is stated in the following lemma.

Lemma 5. *Let A be a general input bitstream of length N and et C be the output bitstream of the RPB encryption module. Then, for a sufficiently large value of N, the first-, second- , and third-order entropies of A and C are equal, that is,*

$$\begin{aligned} H^{(1)}(A) &= H^{(1)}(C), \\ H^{(2)}(A) &= H^{(2)}(C), \\ H^{(3)}(A) &= H^{(3)}(C). \end{aligned} \tag{10}$$

It is straightforward to show that $H^{(1)}(A) = H^{(1)}(C)$ since the number of 0 or 1 in a binary sequence remains the same under the RPB operation. The number of digrams (2-bit subsequence) and trigrams (3-bit subsequence) will fluctuate under the RPB operation due to the rotation of bit blocks. However, we can show that for a sufficiently long sequence, the rise and fall cancel out and the average number of digrams and trigrams remains the same. Thus, the second- and third-order entropies of A and C are still the same. A detailed proof is given in Appendix A.5.

The above lemma demonstrates that RPB encryption does not affect the overall statistical distribution of single bit, 2-bit, and 3-bit subsequences for a sufficiently long input stream A. For orders higher than the partitioned block length of RPB, the entropies of A and C are no longer equal each other. However, it should be noted that the high-order entropies may not be so important in evaluating the statistical properties and redundancy of an information source.

The fact that $H(A) = H(C)$ shows an excellent property of the RPB encryption scheme: the randomness structure of input stream A is retained so that output stream C is statistically as random as A. In terms of information theory, this means that the RPB operation does not increase or decrease the redundancy of output stream C as compared to input stream A.

8. EXPERIMENTS AND PERFORMANCE EVALUATION

Experiments were conducted to evaluate the encryption effect of the joint REC/RPB scheme and reported in this section. We also examine RPB's impact on the statistical randomness of its input A and output C by measuring and comparing entropies of A and C.

8.1. *Experimental setup*

Our experiments were conducted using an H.264 software encoder/decoder. It was based on the reference code (in the C programming language) from the H.264 standard workgroup. We made some slight modifications and used certain optimization techniques (such as the assembly language routine for DCT) to improve its performance. The H.264 baseline profile (BP) was used. A single reference frame was adopted for motion estimation in encoding.

H.264 at the BP level supports the CAVLC entropy coding and 13 Huffman tables are provided in the draft standard. The RHT encryption was implemented with our own software encoder/decoder. Sixteen different tables were constructed for each of the 13 original Huffman tables. When a symbol was input to the entropy coder, one of the 16 corresponding tables were randomly selected to encode that symbol. The RPB encryption was used to process the output of entropy coding afterwards. In our implementation, the temporary output buffer size was set to $B = 32$ bits. The KHS in the RHT encryption and the partition and rotation key sequence in the RPB encryption were generated by repeatedly hashing $s, s+1, s+2, \ldots,$ where s is the initial key value, using the 128-bit MD5 hash function.

8.2. *Encryption effect of joint REC/RPB scheme*

We used a test video clip called "Foreman" from the H.264 workgroup, of the YUV 4 : 2 : 0 format and the CIF size. The first 10 frames were compressed and encrypted using the aforementioned H.264 encoder. The key value was chosen to be 0x246CCA6B103C95. To evaluate the encryption effect, the following 3 experiments were conducted. (1) At the decoder side, the encrypted bitstream was decoded normally *without* applying any decryption algorithm. This test demonstrates the effect of decoding an encrypted bitstream using the normal H.264 decoding process.

(2) At the decoder side, the encrypted bitstream is decrypted using a randomly generated key 0x17460FD05B9EDF. This test simulates the scenario where an adversary attempts an attack by decrypting the bitstream using a randomly picked key.

(3) At the decoder side, the encrypted bitstream is decrypted using the key 0x246CCA6B103C94, which is different from the correct key by one (the last digit being 4 instead of 5). This is to illustrate the extreme case where an attacker was able to discover most parts of the correct key value.

As shown by images given in Figures 6–8, all 3 tests yield totally scrambled, meaningless video content, indicating satisfactory encryption results. Needless to say, decrypting the bitstream using the correct key produces the same image as that of standard H.264 encoding/decoding result.

8.3. *Entropy measurement of bitstreams A and C*

As discussed in Section 7.4, the RPB encryption module does not alter the entropy of input stream A. In this section, we measure the entropies of the coded video bitstream A and the encrypted bitstream C. First, counts of subsequences in A and C are listed in Tables 4~6.

The entropies of order up to 4 are calculated and compared in Table 7. Note that the entropy measure is normalized by dividing the ith entropy by i for the ease of comparison. Several important observations about these empirical statistics are summarized as follows.

(1) Entropies of input stream A are all very close to 1 bit/symbol, which is the entropy of an ideal random binary sequence. This provides a proof that A is a nearly random sequence.

(2) For the test bitstream, the RPB operation has led to a more even distribution of subsequences (of length up to 4) in C as compared to that of A. For instance, the number of 3-bit subsequence “000” in A is 2843 while it is 2588 in C, which is closer to the average number 20000/8 = 2500. As a result, entropies of C are actually higher than that of A and more closer to 1 bit/symbol, which means that C maintains a higher level of randomness than A. This is because A has certain regular patterns, that is, it contains a highly biased number of byte value 0x00 and 0xFF as mentioned in the FIPS 140-1 poker test.

Thus, for an input with some regular structures, the RPB encryption scheme actually increases its randomness as demonstrated by this particular test bitstream. This corroborates our claim that the RPB encryption in general does not weaken the randomness of its output.

9. CONCLUSION

A joint REC/RPB encryption paradigm for efficient multimedia data protection is presented in this work. By exploiting the structure of entropy coder, the proposed scheme demands very low computation and can be easily implemented at negligible cost. In terms of security strength, our scheme remains secure under ciphertext-only attack and known/chosen plaintext attack. We also demonstrate from information theory’s point of view that our scheme does not weaken the statistical randomness of compressed bit stream. Being efficient and secure, our proposed scheme is suitable to encrypt and decrypt multimedia data in highly demanding applications where large amount of data needs to be processed.

Table 4: Counts of 1-bit and 2-bit subsequences in A and C.

	0	1	00	01	10	11
A	10152	9848	5403	4749	4748	5099
C	10152	9848	5057	4920	4919	5103

Table 5: Counts of 3-bit subsequences in A and C.

3-bit	000	001	010	011	100	101	110	111
A	2843	2560	2324	2424	2559	2189	2424	2675
C	2588	2469	2473	2446	2468	2451	2446	2657

Table 6: Counts of 4-bit subsequences in A and C.

4-bit	0000	0001	0010	0011	0100	0101	0110	0111
A	1487	1356	1286	1274	1229	1095	1210	1214
C	1345	1243	1249	1219	1237	1236	1216	1230
4-bit	1000	1001	1010	1011	1100	1101	1110	1111
A	1355	1204	1038	1150	1330	1094	1214	1461
C	1242	1226	1224	1227	1231	1215	1230	1427

Table 7: The entropy values of A and C with the order equal to 1, 2, 3, and 4.

Entropy	$H^{(1)}$	$H^{(2)}/2$	$H^{(3)}/3$	$H^{(4)}/4$
A	0.99983	0.99893	0.99895	0.99826
C	0.99983	0.99990	0.99979	0.99967

APPENDIX

A. PROOF OF LEMMAS

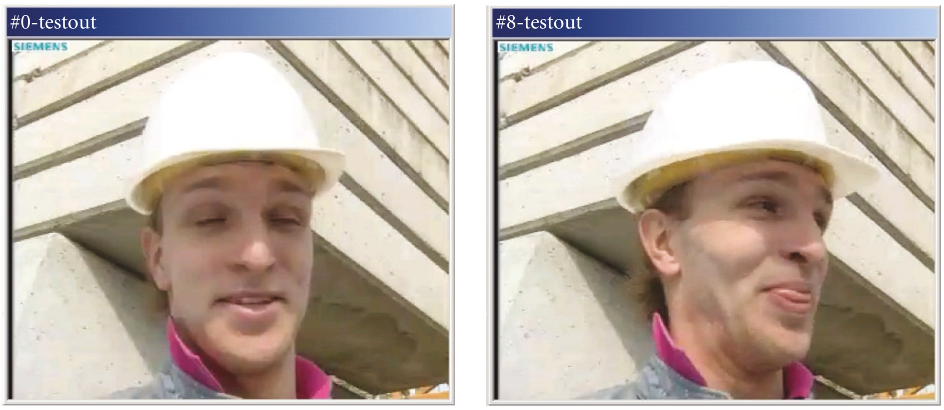
A.1. *Proof of Lemma 1*

Let $A = (a_1 a_2 \cdots a_N)$ be a stream of N bits. Assume B is the maximum block size allowed in, partitioning A. Notice that this restriction implies that any rotation in partition cannot start with bit beyond a_B. Thus, all $R(N)$ possible rotation in partition $RP(A, p, r)$ can be classified into the following B categories:

(1) those starting with a_1;
(2) those starting with a_2;
⋮
(3) finally, those starting with a_B.

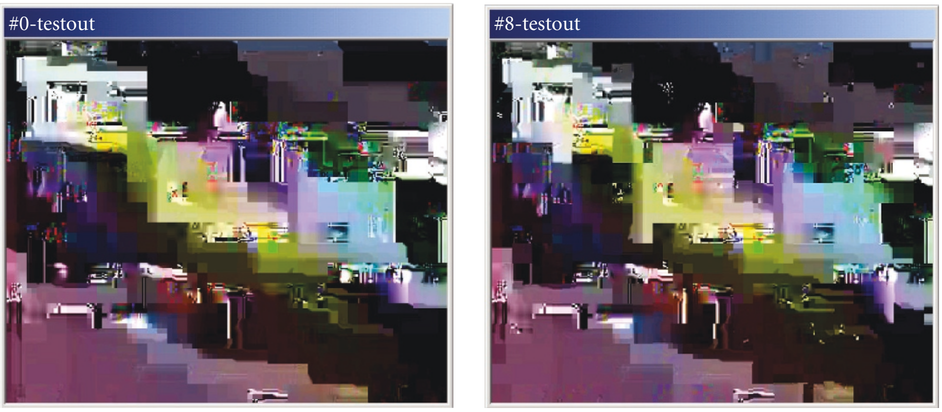
We denote the total number of each category by $R_1(N)$, $R_2(N)$,. . . , $R_B(N)$. Note this classification is mutually exclusive and all inclusive, meaning that any possible resultant bitstream $A' = RP(A, p, r)$ must belong to one and only one of the above categories. Thus, we have

$$R(N) = \sum_{i=1}^{B} R_i(N). \tag{A.1}$$



(a) (b)

Figure 5: Frames 1 and 9 of the test video clip "Foreman".



(a) (b)

Figure 6: Frames 1 and 9 obtained by the normal H.264 decoding process.

Now let us look at each of the above categories. In category (1), a_1 is fixed and we are left with $N - 1$ bits after a_1 which we can freely partition and rotate. Thus, $R_1(N) = R(N - 1)$. In category (2), it must be true that the first $2 \le k \le B$ bits are chosen as a block and a 1-bit left rotation is performed. This is the only way A' can start with a_2. If $k = 2$ (A' starts with a_2a_1), we have $N - 2$ bits left over and total number of possible rotation in partition is clearly $R(N - 2)$. $k = 3$ (A' starts with $a_2a_3a_1$) corresponds to $R(N - 3)$. Finally, for $k = B$ we have the number $R(N - B)$. Notice that this classification with different values of k is again mutually exclusive and all inclusive. Hence, we end up with

$$R_2(N) = \sum_{k=N-B}^{N-2} R(k). \tag{A.2}$$

Continuing the same line of reasoning we have the following equation:

$$R_i(N) = \sum_{k=N-B}^{N-i} R(k). \tag{A.3}$$

Finally, summing up $R_i(N)$, we arrive at

$$\begin{aligned} R(N) &= \sum_{i=1}^{B} R_i(N) \\ &= R(N-1) + \sum_{i=2}^{B} \sum_{k=N-B}^{N-i} R(k) \\ &= R(N-1) + \sum_{k=N-B}^{N-2} (N-1-k)R(k). \end{aligned} \tag{A.4}$$

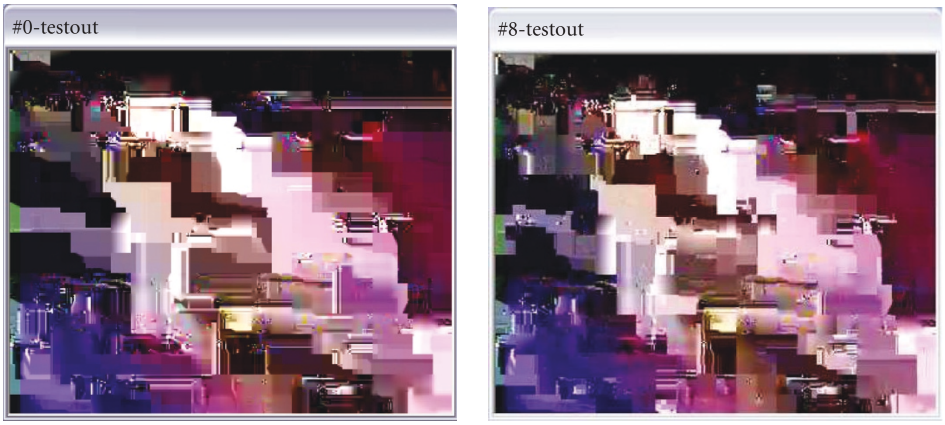
Rearranging the terms of the above equation, we obtain another recursive relationship of $R(N)$:

$$R(N) = 2R(N-1) + \sum_{k=N-B}^{N-3} R(k). \tag{A.5}$$

If we define the following recursive sequence

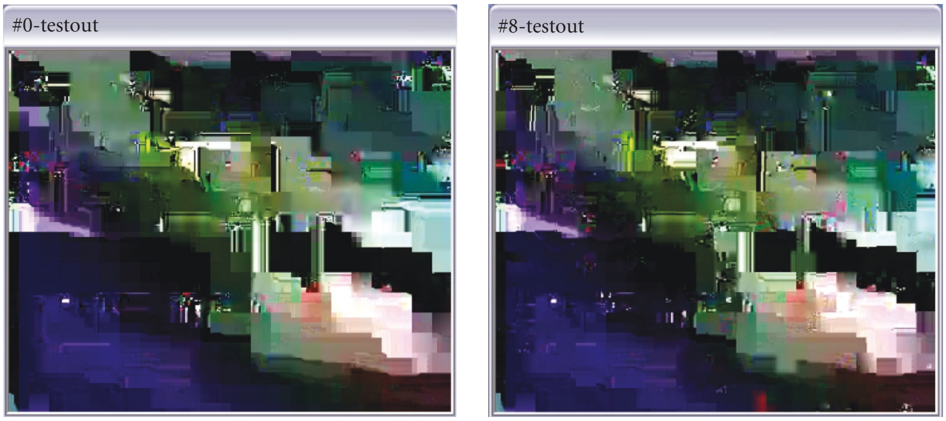
$$S(N) = \begin{cases} 2S(N-1) & N > 0, \\ 1 & N = 0, \end{cases} \tag{A.6}$$

the solution is apparently $S(N) = 2^N$.



(a) (b)

Figure 7: Frames 1 and 9 obtained through decryption using key 0x17460FD05B9EDF.



(a) (b)

Figure 8: Frames 1 and 9 obtained through decryption using key 0x246CCA6B103C94.

From the above definitions of $R(N)$ and $S(N)$, it is clear that if $R(N_0) > S(N_0)$ for some N_0, then $R(N) > S(N)$ for all $N > N_0$. Now, it is readily checked that $R(6) = 65 > S(6) = 64$. Thus, we come to the conclusion that $R(N) > 2^N$ for $N \geq 6$. This completes the proof.

A.2. Proof of Lemma 2

Let $C(N)$ denote the number of different ciphertexts by applying all different RPB operations to A. There are $R(N)$ possible ways to do an RPB operation and the range size is $C(N)$. Thus, by the pigeon hole principle, there must exist a ciphertext C' such that at least $\lceil R(N)/C(N) \rceil$ RPB operations transform A to C'. That is, we have

$$\operatorname{Equiv}(A, C') \geq \lceil R(N)/C(N) \rceil. \qquad \text{(A.7)}$$

Now, let us estimate the size of $C(N)$. Note that an RPB operation only alters the order of bits in plaintext A as a result of random rotation. Since the total number of 0's in ciphertext C' remains the same after any RPB operation, any ciphertext C also contains Z 0's. The total number of N-bit binary sequences containing Z 0's is $\binom{N}{Z}$. However, we claim that $C(N)$ must be strictly less than $\binom{N}{Z}$ due to the upper bound of partitioned block's length $p_i < B$.

Suppose that the first 0 of A occurs at bit position z_1, and the ith 0 at bit position z_i. Because $p_i < B$, the first 0 cannot appear at a bit position after $z_1 + B$ in ciphertext C'. Thus, beginning from bit position $z_1 + B$ to the end, C' can contain at most $Z - 1$ 0's. Likewise, C' can contain at most $Z - i$ 0's after bit position $z_i + B$. This limitation prohibits many bit patterns from being produced as a result of an RPB operation. An accurate analysis of $C(N)$ becomes a quite complicated combinatorial problem since it requires complete knowledge of all z_i's of the particular A. In this study, we take $C(N) < \binom{N}{Z}$ as an upper bound estimate. From Lemma 1, we know that $R(N) > 2^N$. Hence, we conclude

$$\operatorname{Equiv}(A, C') \geq \lceil R(N)/C(N) \rceil > \left\lceil 2^N \Big/ \binom{N}{Z} \right\rceil, \qquad \text{(A.8)}$$

which is (6).

A typical random bitstream A contains half 0's and half 1's on the average. By substituting $N = 2Z$ into the above equations and using Sterling's formula for factorial $n! \approx \sqrt{2\pi n}(n/e)^n$, we obtain

$$C(N) < \binom{2Z}{Z} = \frac{2Z!}{Z!Z!} = \frac{\sqrt{4\pi Z}(2Z/e)^{2Z}}{2\pi Z(Z/e)^{2Z}} = \frac{2^{2Z}}{\sqrt{\pi Z}} = \frac{2^N}{\sqrt{\pi N/2}}. \tag{A.9}$$

Finally, we end up with

$$\text{Equiv}\,(A, C') > 2^N / \frac{2^N}{\sqrt{\pi N/2}} = \sqrt{\pi N/2}. \tag{A.10}$$

This completes the proof.

A.3. *Proof of Lemma 3*

First, we can arbitrarily pick two random N-bit plaintexts, A_1 and A_2, and two N-bit ciphertexts, C_1' and C_2'. By definition of $A(N)$, there are roughly $A(N)$ keys $k_1 \in K_1$ that encrypt A_1 to C_1' and $A(N)$ keys $k_2 \in K_2$ that encrypt A_2 to C_2'. Now, consider the plaintext concatenation $A_1 \parallel A_2$ and ciphertext concatenation $C_1' \parallel C_2'$. It is obvious that a concatenation key $k = k_1 \parallel k_2$ of any $k_1 \in K_1$ and any $k_2 \in K_2$ would encipher the plaintext $A_1 \parallel A_2$ to ciphertext $C_1' \parallel C_2'$. In other words, all such k's are equivalent keys for the $2N$-bit plaintext $A_1 \parallel A_2$, while the average number of a general $2N$-bit plaintext is by definition $A(2N)$. Hence, we end up with

$$A(2N) \geq \#\text{ of such } k = k_1 \parallel k_2 = A^2(N). \tag{A.11}$$

By repeatedly applying the above reasoning to m plaintexts, we have

$$A(mN) \geq A^m(N), \quad m \text{ is an integer.} \tag{A.12}$$

It is a well-known result that a function satisfying the above equation must be of the form c^N. In addition, we have already demonstrated in Lemma 2 that $A(N) > \sqrt{\pi N/2}$ for at least one ciphertext. Therefore, $c > 1$ when N is sufficiently large (otherwise, $A(N) \rightarrow 0$). This completes the proof.

A.4. *Proof of Lemma 4*

Let K_i denote the set of equivalent keys corresponding to each plaintext/ciphertext pair (A_i, C_i), $i = 1, 2, \ldots$. According to Lemma 3, each K_i contains about $A(N) \sim c^N$ keys out of all possible $R(N)$ keys. Denote by X_i the set of possible keys. At the first step, it is clear that $X_1 = K_1$. For each pair (A_i, C_i) checked afterwards, the set of possible keys is the intersection of current set and K_i,

$$X_i = X_{i-1} \cap K_i \tag{A.13}$$

or equivalently

$$X_i = K_1 \cap K_2 \cap \cdots \cap K_i. \tag{A.14}$$

We are asked to find out the index i such that $|X_i|@ = 1$, that is, X_i contains only one element: the correct key.

Instead of directly treating X_i we consider its complementary set $\overline{X}_i = R - X_i$, where R denotes all possible $R(N)$ keys as in Lemma 1. According to set operation law, we have

$$\overline{X}_i = \overline{K_1 \cap K_2 \cap \cdots \cap K_i} = \overline{K}_1 \cup \overline{K}_2 \cup \cdots \cup \overline{K}_i, \tag{A.15}$$

where $\overline{K}_i = R - K_i$ is the complementary set of K_i. Note that $|X_i| = 1$ is equivalent to $|\overline{X}_i| = R(N) - 1$. Since $R(N) > 2^N \gg 1$ we can take $|\overline{X}_i| = R(N)$ which means $\overline{X}_i = R$. Since cryptanalyst cannot arbitrarily manipulate the input stream A, we can assume that each equivalent key set K_i is a random drawing of $A(N)$ keys out of a bin of $R(N)$ keys. Conversely, $\overline{K}_i$ is a random drawing of $R(N) - A(N)$ keys. The attack above can thus be considered a random test. Each step constitutes drawing a random set $\overline{K}_i$ from R and joining the element in $\overline{K}_i$ into the set $\overline{X}_i$. The random test is terminated when $\overline{X}_i = R$, that is, when $\overline{X}_i$ contain all the elements in R. The problem amounts to find the expected number of trials before this random test can terminate.

Apparently, we can see that this problem is a variant of the classic coupon collector problem, where one randomly draws one coupon at a time from a total of N coupons until all N coupons have been collected. It is a well-known fact that the expected number of draws to collect all N coupons is $N \ln N$. The difference is that here the draw size is $R(N) - A(N)$ instead of 1. Due to the randomness of available plaintexts (cryptanalyst cannot manipulate input stream A due to black box structure), all keys in each K_i can be viewed as independently drawn from all $R(N)$ keys with equal probability. Therefore, we can treat one step in our test as an aggregation of $R(N) - A(N)$ tests in the coupon collection problem. The expected number of tests is equal to $R(N) \ln R(N)$ divided by $R(N) - A(N)$, that is,

$$P(N) = \frac{R(N)}{R(N) - A(N)} \ln R(N). \tag{A.16}$$

It is already shown that $R(N) > 2^N$ and $A(N) \sim c^N$ for some constant $1 < c < 2$. For sufficiently large N we have $R(N) \gg A(N)$ and $R(N)/(R(N) - A(N)) \rightarrow 1$. Finally, taking 2^N as a rough estimate of $R(N)$ we end up with

$$P(N) \cong N \ln 2. \tag{A.17}$$

This completes the proof.

A.5. *Proof of Lemma 5*

A general binary sequence s can be treated as output of an binary information source S, which emits 0 and 1 according to its statistical structure. Thus, statistical property of s reflects the statistical structure of the information source S. The nth order entropy for S is defined as follows:

$$H^{(n)}(S) = \sum_d p(d) \frac{1}{\log p(d)}, \tag{A.18}$$

where the summation is over all possible n-bit subsequence d. For a sufficiently long sequence s, the probability of

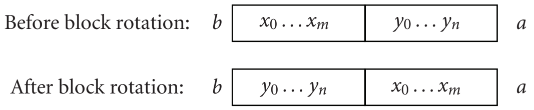


Figure 9: Change of digram under RPB due to bit block rotation.

a particular subsequence d can be calculated as the number of occurrence of d divided by the length of s, that is,

$$p(d) = \frac{\text{\# of occurrence of } d \text{ in } s}{\text{length of } s}. \tag{A.19}$$

For instance, the first- and second-order entropy is computed by

$$\begin{aligned} H^{(1)}(S) &= p(0)\log\frac{1}{p(0)} + p(1)\log\frac{1}{p(1)}, \\ H^{(2)}(S) &= p(00)\log\frac{1}{p(00)} + p(01)\log\frac{1}{p(01)} \\ &\quad + p(10)\log\frac{1}{p(10)} + p(11)\log\frac{1}{p(11)} \end{aligned} \tag{A.20}$$

Now let us look at the input binary sequence A and output binary sequence C as results of RPB encryption. Since RPB operation only reorders certain bit blocks, the total number of 0 and 1 remains the same after passing the RPB unit. It is clear that $p(0)$ and $p(1)$ of output C equal that of input A. Hence, by definition of first-order entropy we have $H^{(1)}(A) = H^{(1)}(C)$.

Things become a little bit complicated for second-order entropy. The number of digrams 00, 01, 10, 11 will vary under RPB operation due to rotation of bit blocks. The change of digrams after a bit block rotation is illustrated in Table 8.

The bit block $x_0 \cdots x_m$ is rotated behind bit block $y_0 \cdots y_n$. b and a are the bits before and after the rotation, either 0 or 1. If we look at all occurrences of digrams, we can find the following changes:

$$bx_0 \longrightarrow by_0, \quad x_m y_0 \longrightarrow y_n x_0, \quad y_n a \longrightarrow x_m a. \tag{A.21}$$

For example, if $(x_0, x_m, y_0, y_n) = (0, 0, 1, 1)$, then the changes are

$$b0 \longrightarrow b1, \quad 01 \longrightarrow 10, \quad 1a \longrightarrow 0a. \tag{A.22}$$

From the changes of digrams, it can be easily seen that there exists a symmetry between x_0, x_m and y_0, y_n. That is, if we exchange the value of x_0 with y_0, and x_m with y_n, then the effect to the change of digrams will be reversed. In the above example, if we take $(x_0, x_m, y_0, y_n) = (1, 1, 0, 0)$, the changes become

$$b1 \longrightarrow b0, \quad 10 \longrightarrow 01, \quad 0a \longrightarrow 1a \tag{A.23}$$

which is clearly the opposite of change corresponding to $(x_0, x_m, y_0, y_n) = (0, 0, 1, 1)$ as in (A.22). The changes of digram for all 16 possible values of the quadruple (x_0, x_m, y_0, y_n) are tabulated below.

Table 8: Change of digram counts under RPB operation.

$x_0 x_m y_0 y_n$	Change of digram
0 0 0 0	No change
1 1 1 1	No change
0 1 0 1	No change
1 0 1 0	No change
0 0 0 1	00→10 $\quad$ $1a$→$0a$
0 1 0 0	10→00 $\quad$ $0a$→$1a$
0 0 1 0	$b0$→$b1$ $\quad$ 01→00
1 0 0 0	$b1$→$b0$ $\quad$ 00→01
0 0 1 1	$b0$→$b1$ $\quad$ 01→10 $\quad$ $1a$→$0a$
1 1 0 0	$b1$→$b0$ $\quad$ 10→01 $\quad$ $0a$→$1a$
0 1 1 0	$b0$→$b1$ $\quad$ 11→00 $\quad$ $0a$→$1a$
1 0 0 1	$b1$→$b0$ $\quad$ 00→11 $\quad$ $1a$→$0a$
0 1 1 1	$b0$→$b1$ $\quad$ 11→10
1 1 0 1	$b1$→$b0$ $\quad$ 10→11
1 0 1 1	01→11 $\quad$ $1a$→$0a$
1 1 1 0	11→01 $\quad$ $0a$→$1a$

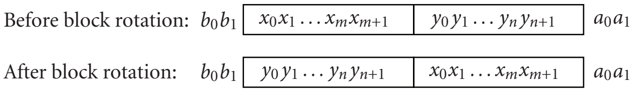

Before block rotation:
b_0b_1
$x_0x_1 \ldots x_m x_{m+1}$
$y_0y_1 \ldots y_n y_{n+1}$
a_0a_1
After block rotation:
b_0b_1
$y_0y_1 \ldots y_n y_{n+1}$
$x_0x_1 \ldots x_m x_{m+1}$
a_0a_1

Figure 10: Change of trigram under RPB due to bit block rotation.

It is clear from this table that the 16 values of (x_0, x_m, y_0, y_n) can be grouped into 6 pairs with opposite change effect and 4 values with no change (because $x_0 = y_0$ and $x_m = y_n$). It is reasonable to claim that these 16 values occur with equal probability due to the following two reasons.

(1) Input stream A, as the result of high performance multimedia compression, can be regarded an almost redundancy-free and nearly random sequence.

(2) In the RPB encryption scheme, the partition size and rotation size are both determined by a pseudorandom sequence. Thus, bit positions of partition boundary and rotation boundary also occur evenly with equal probability.

For a sufficiently long sequence, the rise and fall in number of digrams 00, 01, 10, 11 will cancel out. Hence, we come to the conclusion that in statistical average sense, RPB operation does not change the counts of digrams in a sequence. By probability calculation equation (A.19), the probabilities of digrams for inputs A and C are equal. It follows directly from the definition of second-order entropy that $H^{(2)}(A) = H^{(2)}(C)$.

Now let us go a step further to the third-order entropy. The changes of trigram (3 bit subsequence) due to a bit block rotation is shown in Figure 10.

Again the bit block $x_0x_1 \cdots x_m x_{m+1}$ is rotated behind bit block $y_0y_1 \cdots y_n y_{n+1}$. b_0b_1 and a_0a_1 are two bits before and after the rotation, either 0 or 1. We can note the following changes of trigrams:

$$\begin{aligned} b_0b_1x_0 &\longrightarrow b_0b_1y_0, & b_1x_0x_1 &\longrightarrow b_1y_0y_1,\\ x_mx_{m+1}y_0 &\longrightarrow y_ny_{n+1}x_0, & x_{m+1}y_0y_1 &\longrightarrow y_{n+1}x_0x_1,\\ y_ny_{n+1}a_0 &\longrightarrow x_mx_{m+1}a_0, & y_{n+1}a_0a_1 &\longrightarrow x_{m+1}a_0a_1. \end{aligned} \tag{A.24}$$

Similar to the digram case, we can see that here there is a symmetry between x_i and y_i as well. If we exchange the value of all x_i's with all y_i's, then the effect to change of trigrams will be reversed. Again due to the reason discussed above, we conclude that in statistical average, RPB operation does not change the counts of trigrams in for a sufficiently long sequence. Hence, the third-order entropy of input A and output C are equal, that is, $H^{(3)}(A) = H^{(3)}(C)$. This completes the proof.

REFERENCES

[1] H. Cheng and X. Li, "Partial encryption of compressed images and videos," *IEEE Transactions on Signal Processing*, vol. 48, no. 8, pp. 2439–2451, 2000.

[2] M. V. Droogenbroeck and R. Benedett, "Techniques for a selective encryption of uncompressed and compressed images," in *Proceedings of Advanced Concepts for Intelligent Vision Systems (ACIVS '02)*, Ghent, Belgium, September 2002.

[3] A. Pommer and A. Uhl, "Selective encryption of wavelet-packet encoded image data: efficiency and security," *Multimedia Systems*, vol. 9, no. 3, pp. 279–287, 2003, special issue on Multimedia Security.

[4] M. Podesser, H.-P. Schmidt, and A. Uhl, "Selective bitplane encryption for secure transmission of image data in mobile environments," in *5th Nordic Signal Processing Symposium*, Trondheim, Norway, October 2002.

[5] G. A. Spanos and T. B. Maples, "Performance study of a selective encryption scheme for the security of networked, real-time video," in *Proceedings of the 4th ACM International Conference on Computer Communications and Networks (ICCCN '95)*, pp. 2–10, Las Vegas, Nev, USA, September 1995.

[6] I. Agi and L. Gong, "An empirical study of secure mpeg video transmission," in *Internet Society Symposium on Network and Distributed System Security*, pp. 137–144, San Diego, Calif, USA, February 1996.

[7] L. Tang, "Methods for encrypting and decrypting MPEG video data efficiently," in *Proceedings of the 4th ACM International Multimedia Conference & Exhibition*, pp. 219–229, Boston, Mass, USA, November 1996.

[8] C. Shi and B. Bhargava, "A fast mpeg video encryption algorithm," in *Proceedings of the 6th ACM International Conference on Multimedia*, Bristol, UK, September 1998.

[9] C. Shi, S.-Y. Wang, and B. Bhargava, "Mpeg video encryption in real-time using secret key cryptography," in *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '99)*, Las Vegas, Nev, USA, June 1999.

[10] C.-P. Wu and C.-C. J. Kuo, "Efficient multimedia encryption via entropy codec design," in *Security and Watermarking of Multimedia Contents*, vol. 4314 of *Proceedings of SPIE*, pp. 128–138, San Jose, Calif, USA, January 2001.

[11] J. Meyer and F. Gadegast, "Security mechanisms for multimedia data with the example mpeg-1 video," 1995.

[12] L. Qiao and K. Nahrstedt, "A new algorithm for mpeg video encryption," in *Proceedings of the 1st International Conference on Imaging Science, Systems, and Technology (CISST '97)*, pp. 21–29, Las Vegas, Nev, USA, July 1997.

[13] H.-H. Chu, L. Qiao, and K. Nahrstedt, "Secure multicast protocol with copyright protection," in *Symposium on Electronic Imaging: Science and Technology*, vol. 3657 of *Proceedings of SPIE*, pp. 460–471, San Jose, Calif, USA, January 1999.

[14] A. Slagell, "Known plaintext attack against a permutation based video encryption algorithm," http://citeseer.ist.psu.edu/slagell04knownplaintext.html, 2002.

[15] C.-P. Wu and C.-C. J. Kuo, "Fast encryption methods for audiovisual data confidentiality," in *Symposium on Photonics East, Voice, Video, and Data Communication*, vol. 4209 of *Proceedings of SPIE*, pp. 284–295, Ottawa, Canada, October 2001.

[16] C.-P. Wu and C.-C. J. Kuo, "Design of integrated multimedia compression and encryption systems," *IEEE Transactions on Multimedia*, vol. 7, no. 5, pp. 828–839, 2005.

[17] D. Xie and C.-C. J. Kuo, "Efficient multimedia data encryption based on flexible QM coder," in *Security, Steganography, and Watermarking of Multimedia Contents VI*, vol. 5306 of *Proceedings of SPIE*, pp. 696–704, San Jose, Calif, USA, January 2004.

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

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

[20] H. Kim, J. Wen, and J. D. Villasenor, "Secure arithmetic coding," *IEEE Transactions on Signal Processing*, vol. 55, no. 5, pp. 2263–2272, 2007.

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

[22] D. Xie and C.-C. J. Kuo, "Multimedia data encryption via random rotation in partitioned bit streams," in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS '05)*, vol. 5, pp. 5533–5536, Kobe, Japan, May 2005.

[23] J. Zhou, Z. Liang, Y. Chen, and O. C. Au, "Security analysis of multimedia encryption schemes based on multiple Huffman table," *IEEE Signal Processing Letters*, vol. 14, no. 3, pp. 201–204, 2007.

[24] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, Boca Raton, Fla, USA, 1996.

[25] I. H. Witten and J. G. Cleary, "On the privacy afforded by adaptive text compression," *Computers and Security*, vol. 7, no. 4, pp. 397–408, 1988.

[26] H. A. Bergen and J. M. Hogan, "Chosen plaintext attack on an adaptive arithmetic coding compression algorithm," *Computers and Security*, vol. 12, no. 2, pp. 157–167, 1993.

[27] J. G. Cleary, S. A. Irvine, and I. Rinsma-Melchert, "On the insecurity of arithmetic coding," *Computers and Security*, vol. 14, no. 2, pp. 167–180, 1995.

[28] J. Lim, C. Boyd, and E. Dawson, "Cryptanalysis of adaptive arithmetic coding encryption scheme," in *The 2nd Australasian Conference on Information Security and Privacy*, pp. 216–227, Tokyo, Japan, July 1997.