USC-SIPI REPORT #368

Light-Weight Multimedia Encryption:Algorithms and Performance Analysis

by

Dahua Xie

December 2005

Signal and Image Processing Institute
UNIVERSITY OF SOUTHERN CALIFORNIA
Viterbi School of Engineering
Department of Electrical Engineering-Systems
3740 McClintock Avenue, Suite 400
Los Angeles, CA 90089-2564 U.S.A.

Acknowledgements

I would like to convey my deepest gratitude to advisor Prof. C.-C. Jay Kuo, for his 4 year's guidance and support, for his encouragement during dark times, for his academic spirit to pursue the ultimate answer of each question, for his insatiable dissatisfaction with what has been so far achieved, for his responsive and cooperative working attitude, without all of which this research work is not possible.

I am also greatly indebted to my beloved wife Ying Huo and son Jerrick Yujie Xie, whose love and caring do and will accompany me always.

Contents

List Of Tables

List Of Figures

Abstract

Efficient and secure encryption methods for multimedia content protection are studied in this research. Two new encryption methods, namely the randomized entropy coding (REC) and random rotation in partitioned blocks (RPB), have been proposed. In the REC encryption scheme, an entropy coder is equipped with multiple coding parameters. Encryption is done by randomly choosing one coding parameter according to a pseudo-random sequence to encode consecutive inputs. The RPB encryption scheme exploits the almost redundancy-free characteristics of the coded multimedia bit stream. The bit stream is partitioned into random-sized blocks and a random rotation in each block is performed to achieve encryption. Both the REC and RPB methods have low encryption cost and can be easily implemented in most multimedia coding systems. Both of them achieve the security strength that is between the stream cipher and the block cipher.

This research also investigates distortion-aware multimedia content authentication. A new hash function, called the distance-preserving (DP) hash function is developed to alleviate the hard authentication problem of the traditional hash function. Under the same secret key, two DP hash-values reveal the distance between corresponding preimages. If data have been modified during transmission, this distance provides valuable information about the error and allows distortion evaluation. Properties and security strength of the DP hash function are investigated. A general framework for distortion-aware multimedia authentication based on the DP hash function is outlined.

Chapter 1

Introduction

1.1 Significance of The Research

Recent years have seen a proliferation of emerging multimedia applications, such as digital TV broadcast, on-line video conferencing, distance education networks, etc. Highly advanced multimedia processing technologies and ubiquitous network access are the two most important factors to drive this trend. On one hand, new technologies facilitate the creation, transmission, exchange, and storage of large volume of digitized multimedia data. On the other hand, the need of digital rights protection and management becomes more and more urgent. In particular, the Internet provides a public network that allows illegal distribution of multimedia data much more easily. Thus, multimedia data security, which is the core of digital rights management (DRM) systems, has gained a lot of attention in academia as well as industry.

Data encryption provides a fundamental mechanism to protect the confidentiality of data under distribution. As compared with the encryption of traditional alphanumeric data files, the encryption of multimedia data has encountered several new challenges due to their unique characteristics. First, the size of the multimedia content is often substantially larger than that of the text data. A total encryption approach that enciphers the entire multimedia

content demands a very large amount of computation. Second, most multimedia data seldom exist in raw data format but in compressed data format. An encryption-after-compression approach adds more complexity to the system and increases the software/hardware cost. To apply a separate cryptographic cipher to encrypt compressed multimedia data in a bit-wise fashion is apparently not an ideal solution.

Furthermore, multimedia encryption schemes are also a key technology in the emerging field of wireless and mobile computing. Portable devices such as cellular phones or hand-held devices are heavily constrained in their software/hardware resources and power due to the limitation in their physical size and battery life. Such devices usually cannot afford to perform heavy computation as required by traditional cryptographic ciphers, let alone to equip a dedicated hardware module for the encryption/decryption purpose. In this case, a secure light-weight multimedia encryption algorithm is extremely helpful in saving the system's limited power and resource.

In this research, we study light-weight multimedia encryption techniques to address this problem. In the next section, we examine previous work on multimedia data encryption and establish several design criteria.

1.2 Review of Previous Work

The large size of multimedia data makes it almost impractical to encrypt the entire content. There are numerous research efforts in this field and many multimedia data encryption schemes have been proposed. In the following, we restrict our attention to DCT-based coding methods, since they are the most widely used compression techniques in today's compression standards such as JPEG, MPEG and H.26X. For the encryption of wavelet and quad-tree compressed data, we refer to [1], [2], [3], [4].

To reduce the computational overhead, *selective encryption* is proposed as one solution, where only a specific portion of the data is selected and encrypted. The remaining part of the data is transmitted over the communication channel in plaintext format. Selective encryption is proposed based on the observation that the compressed data are not equally important to the semantic meaning of multimedia content. Some contain the key information that is crucial to the understanding of the multimedia content while others may be less important. For example, motion vectors in the MPEG video are important since they indicate the positions of reference macro-blocks. On the other hand, the least-significant AC coefficients of the DCT transform contain little information since they are usually small in the magnitude. They can be even discarded during decoding without noticeably affecting the quality of decoded video.

One way to implement selective encryption is to select data from the compression domain. Many existing schemes in this category are based on DCT coefficients and motion vectors, since they are generally believed to carry more important semantic information. A security scheme called "Aegis" was developed in [5], which encrypts all I-frames of a MPEG video stream. Intuitively, B and P frames cannot be correctly decoded without help of the I frames. However, Agi and Gong [6] found that video encrypted by Aegis still contains a considerable amount of visually meaningful content, which is largely due to the unencrypted I-macroblocks in B and P frames. Tang [7] proposed to DES-encrypt DC coefficients and use a random permutation instead of the standard zig-zag scan order to scramble AC coefficients. In the Video Encryption Algorithm (VEA) proposed by Shi and Bhargava [8], the 64 most significant sign bits of DCT coefficients and motion vectors in each 16x16 macroblock are encrypted by a symmetric key cipher. An improved version of VEA, called RVEA, was presented in [9]. However, it was observed in [10] that even the DC and the first 8 AC coefficients of all DCT blocks are discarded, it is still feasible to reconstruct an image that contains some meaningful

content. It is also worthwhile to point out that, since the statistical property of the selected portion is altered by the encryption operation, the coding performance of the entropy coder is often impaired. It is a challenging task to select a portion that is critical to the meaning of multimedia content as well as small when compared to the entire data size.

Another way to implement selective encryption is to perform encryption after entropy coding. Qiao and Nahrstedt [11] proposed to split the bit stream into 2 halves (i.e. the *odd* and the *even* halves) according to a random pattern. The ciphertext is formed by $c = odd \oplus even$, followed by $E(even)$, where $\oplus$ denotes the XOR operation and E is an encryption cipher. This approach does provide more security given that the cipher E is secure. However, its computational cost is still high since one half of the entire data needs to be encrypted using a cryptographic cipher.

To summarize, it is difficult for existing selective encryption algorithms to achieve a low encryption cost and a high security at the same time. This is attributed to the following reasons. First, the orthogonal transform such as DCT captures the image energy concentration but not the semantic intelligibility (e.g. the edge and shape cues), which tends to scatter among all frequency domains. Second, selective encryption in the compressed domain changes the statistical property of selected coefficients, which degrades the coding performance of the entropy codec. Third, if selective encryption is performed on the final bit stream after entropy coding, the computational overhead is still high to achieve a high level of security.

1.3 Design Criteria of Multimedia Data Encryption

Based on the review of previous work on multimedia data encryption given in the last section, it is clear that the design of a good multimedia data encryption algorithm is a challenging task. Such an algorithm should meet at least the following requirements.

1. High security

   This is often the most important goal of an encryption algorithm. The security requirement in some applications is very strict. For example, attackers should not be able to extract any meaningful content from the encrypted data without knowing the correct key. However, for some other applications such as movie content protection, a severely degraded quality of the protected content is considered secure and light-weight scrambling techniques suffice to achieve that goal. With the drastic increase of the computing power available to general public and the rapid development of advanced cryptanalysis techniques and tools, the security of any encryption algorithm needs to be evaluated under more stringent conditions. In particular, we require that a good multimedia encryption algorithm should be secure under the ciphertext-only attack, while security against the known-plaintext and chosen-plaintext attacks is highly favored.

2. Low encryption cost

   One main motivation of this research is to cut down the heavy computational cost associated with the traditional total encryption approach. Low cost encryption schemes reduce the load of multimedia processing system and save software/hardware resource, which is particularly important for portable devices. The encryption cost should not exceed a certain portion of the total computation cost of the compression system, including motion compensation, DCT transform, quantization, etc.

3. No harm to compression performance

   Achieving high security at the expense of the coding efficiency is not desirable. A good multimedia encryption scheme should not impair the performance of underlying compression system or cause only very slight degradation.

4. Ease of implementation

   In terms of implementation, it should cost very little to embed the encryption algorithm into the multimedia processing system. If the algorithm is to be added into a portable device, then the software/hardware resource (modification) required should be easy to implement.

5. Compatibility with the compression standard

   The encryption/compression system should be compatible with the compression standard. To be more specific, it should have a configuration in which the encryption scheme is reduced to the standard compression. This provides flexibility to users as they can decide whether to choose an encryption or not according to the security requirements of different applications.

1.4 Contributions of The Research

The following contributions have been made in this dissertation.

- Embedded Encryption via Randomized Entropy Coding

  We propose a new approach for multimedia data confidentiality called Randomized Entropy Coding (REC). The REC method achieves encryption by changing the way an entropy coder works. Instead of always using a fixed entropy coding parameter (setting), REC encodes consecutive inputs by dynamically alternating among multiple entropy coding parameters (settings) according to a statistically random sequence. This embeds the encryption into an entropy coder and eliminates the need of a dedicated encryption module. The REC method does not impair the coding gain of the original data compression scheme. A performance comparison with the traditional block cipher

and stream cipher is conducted. The REC encryption achieves an encryption speed that is comparable with the stream cipher and faster than the block cipher. In terms of security strength, the REC encryption is as secure as the block cipher and the stream cipher under ciphertext-only attack. Its resistance to known/chosen-plaintext attack is better than the stream cipher but generally weaker than the block cipher. The REC approach provides a tradeoff between the two mostly used cryptographic ciphers.

- Bit-Stream-Oriented Encryption via Coded Bit Stream Shuffling

  A general encryption scheme applicable for to compressed multimedia data is presented. We treat the coded bit stream as a random binary 0-1 sequence and alter the order of certain bit blocks as a means of encryption. The scheme is called the random Rotation in Partitioned Blocks (RPB) in that the bit stream is partitioned into random-size blocks followed by a random rotation within each block. We study the RPB operation and show that it can be merged with the algorithm that writes the final bit stream in a multimedia compression system. Thus, the RPB encryption demands very little extra computational cost. Security analysis demonstrates that the RPB scheme is secure under the ciphertext-only attack. For the known/chosen-plaintext attack, its security can be regarded as comparable with the block cipher and is considerably stronger than the stream cipher. It is also established that entropies of the input and the output of the RPB cipher remain the same. Hence, the RPB encryption does not reduce the statistical randomness of its input.

- Security Analysis with Alias Keys

  In the design of REC and RPB encryption schemes, we develop a concept called alias keys, which are different keys that encipher a given plaintext to the same ciphertext. They are useful to the security analysis against the known/chosen-plaintext attack.

For a stream cipher, any single plaintext/ciphertext pair reveals the keystream (but not the key that generates the keystream). For a block cipher, it requires a substantial number of plaintext/ciphertext pairs and a significant amount of computation to mount a successful known/chosen-plaintext attack. But once a key is found, it is almost certainly the correct key. While for the proposed REC and RPB schemes, it is not very hard to find a possible key for a given plaintext/ciphertext pair. But the probability of hitting the correct key is very small with a few number of trials. A cryptanalyst has to examine a huge amount of plaintext/ciphertext pairs to rule out wrong alias keys before the correct key can be determined.

- Distance-Preserving Hash Function for Multimedia Authentication

  A new hash function $Hdist(x, k)$, called the distance-preserving (DP) hash function, is developed. Under the same key k, a certain arithmetic operation on two DP hash outputs reveals a certain distance between inputs. While two DP hash values with different k do not have any special relationship and appear just like regular hash function outputs. When an error occurs, such a DP hash function gives more information between received x' and original x while a regular hash function can only tell $x' \neq x$. The input k is set to a shared secret between the sender and the receiver to control access to the hidden distance information. We consider the construction of a DP hash function based on regular hash functions and analyze its security under the preimage attack and the collision attack. A generalized framework for multimedia content authentication based on the DP hash function is presented. Using the DP hash function to encode the content feature vector, a legitimate user can obtain the distortion information and in turn assess the authenticity degree of the received content.

1.5 Outline of The Dissertation

The rest of the dissertation is organized as follows. The Randomized Entropy Coding (REC) encryption approach is proposed in Chapter 2. The general principle is discussed and three specific encryption schemes are proposed. Performance discussion and security analysis are given and a comparison with traditional cryptographic ciphers is conducted. The random Rotation in Partitioned Blocks (RPB) encryption scheme is presented and studied in Chapter 3. It is a general bit-stream-oriented encryption scheme applicable for any compressed multimedia data. We analyze the computation cost and security strength of RPB scheme and compare it with traditional cryptographic ciphers. The concept of the distance-preserving (DP) hash function is developed and the existence of such function is investigated in Chapter 4. We provide an example to construct the DP hash function, and examine its security as a one-way function. We also propose a generalized framework for multimedia authentication based on the DP hash function. Finally, concluding remarks and directions for future research are given in Chapter 5.

1.6 Summary of Notations

The following notations and abbreviations are used throughout the rest of this thesis.

- $a[i]$: the $i-th$ leftmost bit of binary string a
- $a \parallel b$: 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
- $\lceil x \rceil$: the smallest integer larger than x

- $\{0,1\}^n$: the space of all n-bit binary numbers
- $|\mathbb{S}|$: the cardinality (number of elements) of a set $\mathbb{S}$
- $h(\cdot)$: a cryptographic one-way hash function
- $PRBG$: a cryptographic pseudo-random bit generator

Chapter 2

Multimedia Encryption with Randomized Entropy Coders

2.1 Introduction

Multimedia data encryption is often achieved by cascading a compression module with an encryption module. That is, a cryptographic cipher is applied to the output bit stream of the coding unit. Even though the stream cipher can be implemented efficiently, the encryption cost still poses a considerable amount of overhead on the system due to the large size of coded data. However, this is not the only way to implement encryption. By exploiting the characteristics of the multimedia compression system, it is possible to avoid the use of an extra encryption module yet ensure content confidentiality. In particular, a joint compression/encryption technique that integrates encryption into multimedia data compression is investigated in this chapter.

A new encryption approach called the *Randomized Entropy Coding* (REC) is proposed, in which multiple entropy coding parameters (settings) are dynamically selected according to a statistically random sequence to encode consecutive inputs. Three specific encryption schemes are then developed based on the REC model. They use the Huffman, the arithmetic, and the LZ78 coders as the underlying entropy coders, respectively.

The REC encryption approach has several features. First, its computational cost is low since the encryption process involves only the selection of coding parameters in a random manner. No heavy computation is needed. In terms of implementation, the REC method can be easily included in a multimedia compression system by either software or hardware. Second, it does not affect the coding gain of the original data compression scheme. Third, the REC method is proved to be secure under the ciphertext-only attack. Its security with respect to the known/chosen-plaintext attack is good due to the existence of multiple alias keys, which encrypt a given plaintext to the same ciphertext.

The rest of this chapter is organized as follows. The relationship between compression and encryption is explained in Sec. 2.2. The use of randomized entropy coding (REC) as the encryption cipher is proposed in Sec. 2.3. Three specific examples are given in Sec. 2.4. Then, the computational cost and the security analysis of the proposed REC-based encryption scheme are examined in Sec. 2.5 and Sec. 2.6, respectively. We compare the REC-based encryption with traditional cryptographic ciphers in Sec. 2.7. Experimental results are conducted in Sec. 2.8 to demonstrate the encryption performance. Finally, concluding remarks are given in Sec. 2.9.

2.2 Relationship between Compression and Encryption

In this section, we study the relationship between compression and encryption and show a way to integrate the two into one single module. Without loss of generality, we use a video coder as an example in the following discussion.

A video coder takes a sequence of video frames as input and outputs a binary string of 0's and 1's called the bit stream. The compression algorithm consists mainly of the following stages: motion compensation, DCT transform, quantization, and entropy coding. Motion

compensation eliminates most redundancy between successive video frames by approximating image blocks in the current frame with the closest match in previous frames and encoding only the difference information. DCT transform extracts the image energy in low frequencies (DC coefficients) and high frequencies (AC coefficients). After quantization, many AC coefficients become zero. Intermediate compression results in these stages, such as motion vector, quantized DCT coefficients, are often called quantities (or parameters, coefficients) in the *compression domain.* At the last step of compression, an entropy coder converts these compressed domain quantities to the final bit stream.

All compression operations mentioned above are associated with certain rules, settings and parameters that specify how these operations are performed. Examples include the motion search strategy, quantization threshold, type of entropy coder, etc. Some of them are fixed by compression standards, but certain types of rules and parameters can be customized to cater to particular application requirements. The size and content of the final bit stream varies depending on the particular rules and parameters used in compression. A good example of this is the variable-rate coding where different values of certain parameters, e.g., quantization threshold are used to control the size of output bit stream.

Thus, compression can be regarded as a special type of encryption. We may draw a simple analogy between the two, *i.e.*, the input video, the output bit stream and all compression rules and parameters play the corresponding roles of the plaintext, the ciphertext and the encryption key, respectively. The only difference is that in a compression system, those rules and parameters (the key information) are not kept secret. They are stipulated by standards and become known to the open public. A bit stream encoded under standard rules can always be correctly decoded following the same set of rules. On the other hand, a coding system can be transformed into an encryption system by properly altering some compression rules and

parameters. The particular changes are kept secret and become the encryption key. This provides a possibility to embed encryption into the compression. As a consequence, knowing merely the standard compression rules and parameters does not allow proper decoding of the encrypted bit stream. One has to know those secret changes, i.e., the key information in order to correctly decode the bit stream and obtain the original source data.

Wu and Kuo [12], [10] were the first to explore along this direction. They proposed to integrate encryption with entropy coding. By default, a single fixed statistical model is used throughout the whole entropy encoding process. It is their novel idea to use multiple models (rather than a fixed one) to encode each individual input while the order of the multiple models are kept secret as the key. Specifically, they proposed the Multiple Huffman Table (MHT) scheme and the Multiple State Indices (MSI) scheme for the Huffman and the QM entropy coders, respectively.

2.3 Encryption via Randomized Entropy Coding

Following the discussion in Sec. 2.2, we describe a new encryption approach based on randomized entropy coding in this section. A general entropy coder has certain coding parameters to specify the compression algorithm. Among these coding parameters, some are fixed while others are adjustable (in other words, they can take different values during coding). We can make further distinction between these two types of adjustable parameters.

1. The first type of parameters adjusts their values according to the statistical behavior of input symbols. This often occurs in adaptive coding. The coding parameters are adjusted to better accommodate the change of input statistics, and their values are closely related to the coding efficiency of the entropy coder. One example is the probability estimation in an adaptive QM coder. The probability estimation value is determined

by a state machine, which makes transitions dynamically according to the internal state and the current input.

2. The second type of parameters has nothing to do with coding efficiency. Instead, they are chosen under a general setting of the entropy coder and their choice is just a matter of preference or convenience. The Huffman tree in the Huffman coder serves as an example. Even with a fixed topology of a Huffman tree, we can still use different binary codes to implement the same tree. Since these parameters do not affect the coding efficiency of entropy coders, they are suitable for encryption embedding.

Let us make a formal definition below.

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

1. *It can assume different (often adjustable) values in entropy coding.*

2. *The use of different values of this parameter will lead to different coding result.*

We use the word "equivalent" to emphasize the fact that an ECP can take different values freely and the choice does not have an impact on the efficiency of entropy coding. By default, an entropy coder uses fixed values of ECP to encode all inputs throughout the entire compression process. To design a joint compression/encryption scheme, our idea is to use different values of ECP during the entropy coding. First, we choose an ECP and construct a pool containing multiple ECP values. Then, on each individual input, a particular ECP value is chosen from the pool according to a random sequence and used to encode that input. Since the final bit stream is a cumulation of successive coding outputs, its content varies depending on the particular ECP values used. Thus, we come up with a general encryption

scheme that can be integrated into entropy coder with ECP. This is called the *Randomized Entropy Coding* (REC) approach.

The random sequence that specifies the selection of particular ECP values becomes the encryption key of the REC approach. This sequence is given the name *key hopping sequence* (KHS) since the way our REC works is similar to a frequency hopping communication system. That is, the entropy coder alternates among different ECP values just as the communication frequency hops in different channels according to the input of a random sequence. Apparently, the property of KHS is of utmost importance to the security of the REC-based encryption scheme. One has to be cautious in the design of good KHS to achieve a high level of security.

Let us discuss several desirable properties of a KHS. It is important to realize that the REC approach can be viewed as a series of successive random tests. In each step, an ECP is randomly chosen to encode the current input. Thus, the first requirement is that the KHS should be indistinguishable from a truly random sequence statistically. A cryptanalyst cannot differentiate it from a truly random sequence based on statistical properties such as the mean, the variance, and the distribution of the run length, etc. Second, successive bits in a KHS should be statistically independent. This is because it is always prudent to assume that a cryptanalyst might be able to obtain a fragment of the KHS being used. If this is the case, the statistical independence between successive bits prohibits the cryptanalyst from gaining any useful information about other parts of the KHS. More formally, these two requirements can be formulated as follows.

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

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

In terms of cryptography, the above two conditions are recognized as the *polynomial-time statistical test* and the *next-bit test*, respectively [13]. It is well known that a sequence that meets these two tests is defined as a *pseudo-random bit sequence* and such a sequence can be generated by a *pseudo-random bit generator* (PRBG). A PRBG can be generally considered a finite state machine. Its input is a relatively short binary sequence called the *seed*, which drives the PRBG to output a long pseudo-random sequence.

In the proposed REC approach, we use a PRBG to generate the KHS. The details of the REC encryption approach are presented as follows.

Randomized Entropy Coding (REC)

- *REC set-up*

  1. ECP pool

     For the entropy coder of interest, select an ECP and construct a ECP pool containing n ECP values, $E = \{e_i,\ i = 1, 2, \ldots, n\}$.

  2. KHS generator

     Choose a cryptographically good PRBG as the KHS generator. Generate a random number s as the seed input to PRBG, the output pseudo-random sequence z is used as KHS.

- *REC encryption*

  At the very beginning, the bit stream B_0 is set to the empty string. For each input p_i to the entropy coder, do the following two steps.

1. ECP selection

   Select a particular ECP value e_i from E using a proper number of bits z_i from KHS.

2. Encoding one input

   Use e_i from the above step to encode p_i and produce the output bit stream, which can be written as

$$B_i = Encode(B_{i-1},\ p_i,\ e_i) \tag{2.1}$$

3. Repeat the above two steps until all inputs have been encoded.

The only thing kept secret is the seed s that drives the KHS generation and serves as the key of the REC-based encryption scheme. All other information, such as the entropy coding algorithm, the ECP pool and the PRBG algorithm, can be available to the public. The decryption process is straightforward. The secret key s can be used to generate the same KHS as that in the encryption process, which allows the decoder to recover all plaintext symbols p_i from the encrypted bit stream.

Several important observations about the statistical behavior of plaintext and properties of REC encryption are given below, for which we do not offer rigorous proofs but heuristic explanations.

Proposition 1 (Weak correlation plaintext) *The input symbols to the REC (namely, the compressed domain quantities) have weak or no statistical correlation between them.*

There are compression operations done before entropy coding (e.g. transforms, motion estimation, quantization, etc.) to remove spatial and temporal redundancy contained in the raw video data as much as possible. As a result, the compressed domain quantities before the entropy coding stage should have weak or even no statistical correlations.

Proposition 2 (Error propagation effect) *If a single bit error occurs at any position in the KHS, then the decryption result afterwards will be different from the correct plaintext.*

We see from Eq. (2.1) that the ciphertext is a cumulative function of all previously used ECP values. Hence, there is a stringent synchronization issue. Correct decryption counts on strictly following the same KHS used in encryption. Suppose that a single bit error in the KHS leads to a wrong ECP value to be used at a certain point. After this point, synchronization is lost and all following decryption will in general give wrong results.

Another important characteristics of the REC encryption is that for given ciphertext B and input plaintext symbol p, there may exist more than one ECP values that would yield the same encryption result in Eq. (2.1). Mathematically, this can be written as follows.

Definition 2 *The alias coding parameter (ACP) set A_p is the largest subset of the ECP pool E that satisfies the following condition:*

$$Encode(B, p, a_i) = Encode(B, p, a_j), \quad \forall a_i, a_j \in A_p \tag{2.2}$$

where B denotes a cipher text and p is an input plaintext symbol. The alias degree $A(p)$ of symbol p is defined as the cardinality of set A_p.

The alias degree is obviously plaintext-dependent. Each plaintext symbol has its own ACP set. The above characteristics play a key role in the security strength of the REC-based encryption scheme. We will provide a detailed security analysis in Sec. 2.6.

2.4 Examples of REC-based Ciphers

Three specific REC-based encryption schemes are presented in this section. They are associated with the well known Huffman, arithmetic and Lempel-Ziv entropy coders, respectively.

2.4.1 RHT (Randomized Huffman Tree) Scheme

Huffman coding is one of the most widely used entropy coder in most image/video compression system. We can construct an REC-based encryption scheme based on the Huffman coder. The Huffman tree is a good ECP since the same tree can be associated with different Huffman codes. In the proposed Randomized Huffman Tree (RHT) encryption scheme, multiple Huffman codes are constructed that correspond to the same Huffman tree. This can be easily done using a technique called the Huffman tree mutation process [10]. Then, a particular code is chosen to encode the plaintext input according to the KHS.

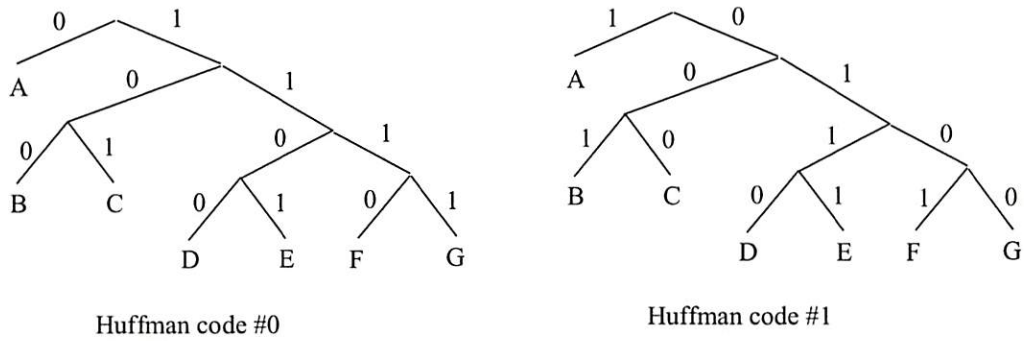
The RHT scheme is actually similar to the improved MHT scheme as given in [10], where the output of a stream cipher is used as the KHS and a segment of KHS is repeatedly used for a certain number of times before moving to a new segment to reduce the computational cost of KHS generation. Since the output of a stream cipher can be regarded as a pseudo-random sequence, the RHT scheme and the improved MHT scheme are basically the same. We mention the RHT scheme here only to show that the improved MHT scheme in [10] can be unified under the REC.

Because Huffman coding is the most-widely used and most-easily understood entropy coder, we give an example of RHT encryption scheme below to illustrate the REC idea.

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 Fig. 2.1, are constructed to encode these 7 symbols. Note that the topology of two Huffman trees is the same so the code length of each symbol is identical, although the code values are different. A sample input plaintext $P = ACDABEFG$ is encrypted using two keys

$$k_1 = 00000000, \quad k_2 = 10011010,$$

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 input. The key value and the corresponding ciphertext are shown in Fig. 2.2 with different ciphertext bits highlighted by the blue color. It is clear that the difference depends on the particular key value chosen.



Huffman code #0

Huffman code #1

Figure 2.1: Two Huffman trees with the same topology

Plaintext	Key	Ciphertext
ACDABEFG	00000000	010111000100110111101111
	10011010	110111001001110101011111

Figure 2.2: RHT encryption using two different keys

Assume that plaintext P is encrypted using key k_2 with the ciphertext as shown in the 2nd row of Fig. 2.2. Next, we study the effect of the RHT decryption with 3 keys as shown in Fig. 2.3. The first key is the correct one so that it recovers plaintext P successfully. The second key is the all-0 key which emulates the situation where the receiver decodes the RHT-encrypted ciphertext using the standard Huffman decoding procedure. The decoded result is totally different from the correct plaintext P. Decryption using the third key demonstrates the error propagation effect. The third key is different from the correct key only in a single bit;

namely, at the location of the 3rd bit. The first 2 plaintext symbols are decrypted correctly. However, the decryption result is totally wrong starting from the 3rd plaintext symbol.

Ciphertext	Key	Plaintext
110111001001110101011111	10011010	ACDABEFG
	00000000	EDBFCAG
	10111010	ACAABAEA

Figure 2.3: RHT decryption using three different keys

2.4.2 RACCI (Randomized Arithmetic Coding Convention Interleaving) Scheme

The binary arithmetic coder is another popular entropy coder that is widely used in multimedia compression. There was previous work using adaptive arithmetic coding as a means of encryption. However, the proposed schemes are not satisfactory in terms of security and complexity. Readers are referred to [14], [15], [16], [17] for discussion of these schemes and their security analysis. Here, we propose a new encryption scheme based on arithmetic coding. It is called the Randomized Arithmetic Coding Convention Interleaving (RACCI) encryption. As suggested by the name, the ECP chosen for this scheme is the coding convention in binary arithmetic coders. Simply speaking, arithmetic coding is a process of repeatedly subdividing an interval, and any point in the current interval can be used to represent the bit stream. For a given bit stream, it is possible to encode any symbol by any convention, provided that the target user knows the order of the conventions to be used. In other words, we use the coding convention as the ECP.

To be more precise, there are two possible symbol orderings (i.e. either the LPS subinterval above the MPS subinterval or the MPS subinterval above the LPS subinterval) and two possible code stream conventions (i.e. the code stream points to either the bottom or the

top of an interval) for binary arithmetic coders. Thus, we are led to a total of four possible coding conventions as given below, where C denotes the encoding code stream and A is the updating interval, Q_e is the probability of the least probable symbol. Figure 2.4 on the next page 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

Although there are four possible coding conventions above, we adopt only conventions (a) and (b) in the proposed RACCI scheme for two reasons. First, there is a relationship between the code streams of conventions (a) and (c). Even though the two coding procedures are different in their implementations, the difference between them is actually the remaining probability interval A. There is a similar relationship between the code streams of conventions (b) and (d). Second, since the KHS is a binary bitstream, there is no advantage to use more than two conventions. The use of two coding conventions is sufficient to provide a high security level. The proposed RACCI encryption scheme is described below.

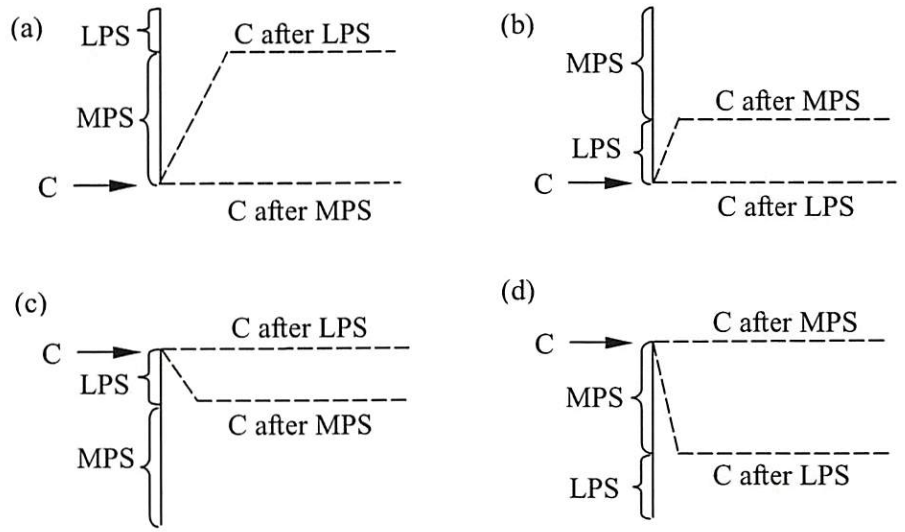


Figure 2.4: Four coding conventions of arithmetic coding

Summary of the RACCI Encryption Algorithm

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

2. Let z be the KHS output from the KHS generator.

3. For the ith incoming symbol

   if $z[i] = 0$, use convention (a) to encode the symbol.

   if $z[i] = 1$, use convention (b) to encode the symbol.

4. Repeat Steps 3) until all symbols are encoded.

2.4.3 RDT (Randomized Dictionary Table) Scheme

Ziv and Lempel proposed two sequential data compression algorithms [18], [19] in 1977 and 1978, which were known as LZ77 and LZ78, respectively. These algorithms form the basis of a few lossless text compression methods, including LZW, LZSS, arc, zip, among many others. The LZ coders belong to the category of dictionary coders, where the future source output is encoded via maximum-length matching against a dictionary that contains the most recent outputs. It is specifically suitable for coding applications where the entropy of the target data is not known in advance or dynamically changing.

The LZ77 scheme maintains a fixed-size buffer that keeps track of the most recent n bytes of the source output. The future source output is scanned and compared against this n-byte buffer. When the maximum-length match in the buffer is obtained, it outputs the pair (*offset, length*) indicating the position of the matching phrase in the buffer, and the length of matching.

The LZ78 scheme works differently by entering previously-seen strings in a dictionary table. The table initially contains all characters in the alphabet set. The future output is scanned to find the longest re-occurrence of strings in the dictionary. The coder then outputs the dictionary index of that particular string A and update the dictionary by adding Ab as a new entry, where b is the next character following string A.

In practice, LZ-based coders are the most popular lossless data coding schemes for the compression of alpha-numeric files due to its superior compression performance and fast speed. The most time-consuming operation in LZ-based coders is obviously the search for the longest match. A complete linear search over the entire dictionary requires (dictionary size) $\times$ (string length) times of comparisons, which is too large to be practical.

Several techniques have been developed to speed up the search in real world implementations, *e.g.* the use of a hash table. A hash table has a linear index array while each entry of this array points to a linked list (in some cases also linear array), which contains the information related to the index. As the LZ coder reads in M characters, it will compute a hash key of these M characters, which is used to index the hash table. For LZ77, The linked list at the indexed entry contains positions in the buffer where the M-character pattern appears. For LZ78, each node in the linked list is a 2-tuple (*string, index*) where *string* is a string starting with the M-character pattern and *index* is the index of that particular string in the dictionary table.

Given the above sketch about how LZ coders work, it appears difficult to construct an REC encryption scheme based on the LZ77 coder since the dictionary, which contains the most recent n bytes of the source output, is a fixed buffer that does not allow any change.

However, for the LZ78 coder, we propose an encryption algorithm called the Randomized Dictionary Table (RDT) scheme. We construct more than one dictionary tables with a different entry order as multiple ECP values. The following two techniques are used to achieve a random entry order in different dictionary tables.

1. *random initial dictionary*

   First, we arrange the initial dictionary (the basic alphabet) into a secret order other

than the default order in the standard LZ78 scheme. This secret order can be set according to a random permutation on the alphabet.

2. *random dictionary insertion*

   New entries are not added into the dictionary by the sequential order as done in the standard LZ78 scheme. Instead, we select a random position in the dictionary to insert the new entry. The original entry at the target position is moved to the end of dictionary by modifying its index to the current dictionary size plus one. In the case where the dictionary is full, this is equivalent to the replacement of a random one in the dictionary by the new entry.

In each LZ78 encoding step, we randomly choose a dictionary table according to the KHS. Also, a new entry will be added in all dictionary tables. To maximize the alias degree, we randomly split all dictionary tables into two halves and insert the new entry in the same random position in each half. The choice of splitting halves can be specified by the lexigraphical order of elements in the first half. For instance, if we have 4 tables, then there are $C_4^2/2 = 3$ possible ways for splitting, which can be unambiguously ordered as $\{1,2\},\{1,3\},\{1,4\}$ with the remaining halves equal to $\{3,4\},\{2,4\},\{2,3\}$, respectively. As a result of this random splitting and insertion, an entry will have the same index in a random half of all dictionaries in the long run. This maximizes the alias degree for each entry. The overall RDT scheme is summarized as follows.

Summary of the RDT Encryption Algorithm

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

2. Generate E initial dictionary tables with a different entry order, numbered from 0 to $E-1$.

3. Let z be the KHS output of the KHS generator.

4. Generate a random number $0 \leq d < E$. Choose a dictionary table according to d and use it to perform one LZ78 encoding step.

5. Do the following to add the new entry *str* to all dictionary tables.

   - Generate a random number $0 \leq c < \binom{E}{E/2}/2$. Choose a splitting halves according to c by the lexigraphical order as described above.
   - Generate two random numbers r_1 and r_2 in the range $0 \sim N$.
   - Add the new entry *str* to position r_1 of the dictionary tables in the first half and add the new entry *str* to position r_2 of the dictionary tables in the second half.

6. Repeat Steps (3)-(5) until all data are compressed and encrypted.

Note that parameter N in above denotes the maximum dictionary size. Furthermore, to generate a random integer between $0 \sim N$ from a random sequence z, we can first take the right-most $q = \lceil log_2 N \rceil$ bits of z, which is set to number t, and then scale t to the range $0 \sim N$ by performing $N \times t \gg q$, where $\gg$ is the right shift operation.

2.5 Computation and Storage Cost Analysis

For the second half of this chapter, we will evaluate the performance of the proposed REC-based encryption technique such as computational and storage costs and the security strength in this and the next sections. We will also compare the proposed scheme with traditional block and stream ciphers in Sec. 2.7.

The computational/storage overhead of the REC-based encryption schemes consists of three parts: (1) the cost to set up the initial REC, (2) the cost to generate the KHS, and (3) the cost to allow the entropy coder to have multiple ECP values. They are analyzed separately below.

1. *Set-up cost*

   The primary cost to set up the REC-based cipher is the preparation of the ECP pool. This includes constructing multiple ECP values and the memory to store them. Usually, these values can be organized in a linear storage space so that they can be easily accessed by the entropy coder in later operations. This initial cost is usually low since ECP values take a small memory space and storing them in a linear storage is easy. Take the RHT encryption as an example. Suppose that each Huffman tree has 12 entries and it takes 5 bytes to store one entry (*i.e.*, 1 byte for length and 4 bytes for the code value). Then, a single Huffman tree demands 60 bytes. If 16 different trees are used, the memory requirement is only 960 bytes. Besides, this initial set-up cost is a one-time cost which can be amortized over a large amount of encryption that follows.

2. *KHS generation cost*

   This is the major computational overhead of the REC-based encryption scheme since the length of KHS required to encrypt all inputs is proportional to the length of the plaintext data. The pseudo-random number and its generation are an important topic in cryptography, and there exist many techniques to generate pseudo-random sequences. These methods can be classified into three categories: (1) those based on symmetric key encryption including the block and stream ciphers, (2) those based on public key encryption such as RSA, and (3) those based on the one-way hash function. As far as the computational speed is concerned, the stream cipher is the fastest one. The speed

of the block cipher and the one-way hash function are roughly the same but slower than the stream cipher. The public key technique is the slowest since it demands expensive computation such as exponentiation in a group. Thus, the KHS generation cost depends on the specific cryptographic technique used to generate the pseudo-random sequence.

3. *Cost for an entropy coder to take multiple ECP values*

   The entropy coder has to be configured flexibly so that it can take different ECP values. If the entropy coder is implemented in software, this can be easily done by adding a variable index (depending on KHS) to the base address. Usually, it takes no more than 2 to 3 instructions to accomplish this task. If the entropy coder is implemented by hardware, this cost translates to several kilobytes of memory to store multiple ECP values plus a couple of multiplexer and control logic to index into them. This cost is usually much lower as compared to the KHS generation cost.

In summary, in the proposed REC-based approach, the set up cost and the cost of adapting the entropy coder to multiple ECP values are relatively small. The primary computational overhead comes from the KHS generation, which depends on the underlying pseudo-random sequence generation technique.

2.6 Security Analysis

2.6.1 Ciphertext-Only Attack

In the ciphertext-only attack, a cryptanalyst has only certain ciphertext available for study and the goal is to decrypt the corresponding plaintext and/or recover the encryption key. Since the plaintext is not known, this is the least favorable situation for a cryptanalyst. Due

to pseudo-randomness of the KHS, the REC-based scheme is immune to the ciphertext-only attack, which is formally stated below.

Theorem 1 *Under the ciphertext-only attack, breaking the REC-based scheme is at least as difficult as breaking the underlying KHS generation algorithm, for which a brute-force attack requires a computational complexity of* $\min(2^r, E^L)$*, where* r *is the bit length of seed* s*,* E *is the cardinality of ECP pool, and* L *is the length of the corresponding plaintext.*

Proof : By "breaking the REC model", we mean that, based on a given REC-encrypted ciphertext, an algorithm can be devised to yield the corresponding plaintext as its output. If such an algorithm exists, we conclude that it must know the KHS sequence to compute the correct plaintext due to the following arguments.

In a ciphertext-only attack, the only information available to the cryptanalyst is the bit stream of a given ciphertext. According to Eq. (2.1), the ciphertext bit stream is a function of all ECP values e_i used. Thus, the algorithm must depend on the knowledge of correct e_i used in each encoding step to compute the plaintext. We can further argue that the algorithm must know the corresponding KHS in order to find out each correct e_i.

Starting from initial ciphertext B_0, it is equally likely for e_0 to take a value in the ECP pool since nothing is known about the first plaintext symbol p_0 at this time. Thus, the algorithm has to know the KHS z_0 to choose the correct e_0 and decrypt p_0. Next, let us assume plaintext symbols $p_0, p_1 \ldots$ up to p_k have been decrypted. Since Proposition 1 says that the next plaintext p_{k+1} does not depend on p_i, $i = 0, 1, \ldots k$, the previously decrypted plaintext symbols do not help in decrypting the future plaintext due to the weak or no correlation property. This is exactly the same as to decrypt the first plaintext symbol p_0. Since no additional hint about p_{k+1} is available, e_{k+1} can be any value in the ECP pool with

an equal probability. Thus, the algorithm has to know KHS z_{k+1} to determine the correct e_{k+1} and in turn decrypt p_{k+1}.

In summary, all decryption steps are independent of each other. To compute the correct plaintext, the algorithm has to know the corresponding KHS sequence. Hence, if an algorithm exists to decrypt an REC-encrypted ciphertex, such an algorithm can also be used to give a particular pseudo-random sequence produced by the underlying KHS generation algorithm. In this sense, we say that breaking the REC-based scheme is at least as difficult as breaking the underlying KHS generation algorithm.

Next, let us examine the complexity of a brute-force attack. There are two approaches to attack the KHS of a given ciphertext. The first method is to try all possible KHS to see whether the decrypted plaintext is semantically meaningful. The second strategy is a brute-force search for the secret seed s since the underlying KHS generation algorithm is available to the public. A random seed value is picked to calculate a KHS, which is used to decrypt the ciphertext until a meaningful result is encountered. For the first method, the length of the plaintext is L so that it takes L decryption steps to recover the plaintext. As analyzed earlier, all these steps are independent of each other. In each step, there are E possible choices that are equally likely. Therefore, the entire search space is $O(E^L)$. For the second method, since the seed is a random r-bit binary number, all r-bit numbers occur with an equal likelihood. In other words, the seed may be produced as a result of tossing an unbiased coin r times, recording 0 for the head and 1 for the tail. Thus, the cryptanalyst has no better strategy other than searching all 2^r possible r-bit binary numbers. The entire search space in this case is $O(2^r)$. Thus, it is concluded that the computational complexity of a brute-force attack to REC-based schemes is $\min(2^r, E^L)$. This completes the proof. ■

2.6.2 Parsing Attack

The parsing attack is a particular attack for the RHT encryption scheme. In standard Huffman decoding, there is only one possible way to decode the Huffman-coded bit stream due to the unique decodability property of the Huffman code. However, to decode an RHT-encrypted bit stream, the correct KHS is needed. Thus, a proper Huffman table is used to parse the bit stream at each step. If a wrong KHS is used, we may end up with several remaining bits that cannot be parsed exactly at the end. A cryptanalyst can exploit this characteristic to determine whether a KHS is correct or not.

For a general N-bit RHT-encrypted bit stream C, a particular parsing is called a *viable* parsing scheme if C can be exactly parsed without leaving any bits at the end. The corresponding KHS associated with a viable parsing scheme is called the *viable* KHS. Decoding C using the correct KHS is a viable parsing scheme. The question is whether there exist other viable parsing schemes that can parse C exactly. If the answer is positive, the next question is how many of them? To address this problem, we first define the following concept.

Definition 3 *Let C be an N-bit RHT-encrypted bit stream. The viable k-parsing scheme of C is defined as a parsing scheme that can exactly parse the first k bits of C. We use $V(k)$ to denote the total number of viable k-parsing schemes of C. If $V(k) \geq 1$, we call k a parsable length,* i.e., *there exists at least one vialbe k-parsing scheme.*

By parsing bit stream C using all possible ways and recording the parsed code length, a cryptanalyst can gradually compute values of all $V(k)$ for $1 \leq k \leq N$. The detailed parsing attack is described below:

Parsing Attack

1. To initialize, we set all $V(k) = 0$, $1 \leq k \leq N$ and VP to the empty set $\emptyset$.

2. Parse C using each Huffman table and record the parsed code length k_i, $i = 1, 2, 3 \ldots$ respectively.

   Add all k_i to VP. $V(k_i) = 1$, $k_i \in VP$.

3. Sort the set VP in ascending order.

   For each element $p \in VP$

   Parse C from the $(p+1)$-th bit using each Huffman table and record the parsed code length k_i respectively.

   Remove p from VP and add $p + k_i$ to VP for all k_i.

   $V(p + k_i) = V(p + k_i) + V(p)$ for all k_i

4. Repeat step 3 until the bit stream C is parsed.

After running the above algorithm, a cryptanalyst can obtain all parsable lengths n and the value of corresponding $V(n)$. In particular, $V(N)$, the number of viable parsings for the entire bit stream C, can be computed. Clearly, the threat level of the parsing attack to RHT encryption depends on the size of $V(N)$. If $V(N)$ is relatively small (say, polynomial with N), then parsing attack poses an immediate threat to RHT encryption. We study the size of $V(N)$ in the following. First, the following properties of $V(N)$ will be established.

Lemma 1 *Let C be a general N-bit RHT-encrypted bit stream. The total number of viable parsing schemes as defined in Def. 3 satisfies the following properties.*

1. *For any consecutive $L+1$ integers between 1 and N, $1 \leq s, s+1, \ldots s+L \leq N$, at least one of them is a parsable length. That is, there exists at least one integer $s + \alpha$ such that*

$$V(s + \alpha) \leq 1, \quad 0 \leq \alpha \leq L, \tag{2.3}$$

   where L is the maximal code length of all Huffman tables used in RHT encryption.

2. *Let c_i denote the i-th bit of C. For any integer $1 < k \leq N$, we have the following recursive equation:*

$$V(k) = \sum_{\beta_i} V(k - \beta_i), \tag{2.4}$$

where $c_{k-\beta_i+1}\, c_{k-\beta_i+2} \cdots c_k$ is a β_i-bit Huffman code in one of all Huffman tables used in RHT encryption.

3. *For any integer $1 < k \leq N$, let $S = \{s, s+1, \ldots s+L\}$, $s + L \leq k$ be any consecutive $L+1$ integers. $V(k)$ can be represented as a weighted sum of $V(i)$ with indices i in S; namely,*

$$V(k) = \sum_{i} a_i V(i),\ i \in S, \tag{2.5}$$

where a_i's are integer coefficients.

The key to prove the above Lemma is the observation that the gap between two consecutive viable parsing positions must be less than $L+1$ because L is the maximal code length of all Huffman tables. By tracing the viable parsing positions backward, we can derive the recursive equation of $V(N)$ 2.4 and 2.5. A detailed proof is given in Appendix A.1.

Next, we examine the size of $V(N)$ with respect to bit stream length N. Equations (2.4) and (2.5) suggest that the size of $R(N)$ will grow very quickly for large value of N. It is a well-known fact that a function satisfying such a recursive relationship has a form of $c\,a^N$ with constant $c > 0$ and $a > 1$. Thus, we expect $V(N)$ to have an exponential rate of growth with respect to N. However, the code length β_i in (2.4) and coefficients a_i in (2.5) are not constant values since they vary with the particular bit stream C and all Huffman tables. Thus, it is not easy to derive a determinate formula of $V(N)$ as a function of N. We establish a lower bound of the size of $R(N)$ below.

Because there are more than one Huffman tables used in RHT encryption, there are at least two last parsed codes with different length β_i. Their code lengths are denoted by β_1 and β_2. Then, according to (2.4), we have

$$V(k) \geq V(k-\beta_1) + V(k-\beta_2),\ 1 \leq \beta_1, \beta_2 \leq L.$$

According to the definition of $V(k)$, it should be a non-decreasing function of k. The longer the bit stream, the larger the number of viable parsing schemes. Since $\beta_1, \beta_2 \leq L$, we have $V(k-\beta_1) \geq V(k-L)$ and $V(k-\beta_2) \geq V(k-L)$. Therefore, we get

$$V(k) > V(k-L) + V(k-L) = 2V(k-L).$$

The solution of $A(k) = 2A(k-L)$ is clearly $A(k) = 2^{k/L}$. Thus, we have $V(k) > 2^{k/L}$ and, for the N-bit stream C, the number of viable parsing schemes satisfies

$$V(N) > 2^{N/L}, \tag{2.6}$$

where L is the maximal code length of all Huffman tables used in RHT encryption.

Finally, we use the following example to illustrate the number of viable parsing schemes, $V(N)$. The test bit stream is the following 24-bit binary sequence

110111001001110101011111

as shown in Fig. 2.2, which is the encryption result of plaintext input "ACDABEFG" under the KHS "10011010" using two Huffman tables in Fig. 2.1. The number $V(k)$ of viable

parsing schemes are obtained for length $1 \leq n \leq 24$ by running the parsing attack algorithm on this bit stream.

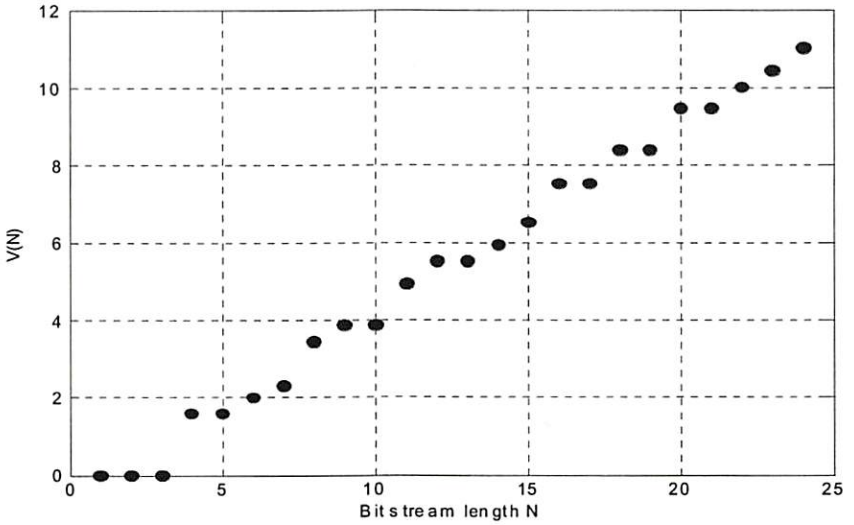


Figure 2.5: The number of viable parsing schemes for a 24-bit stream, where the Y-axis is in the log scale with base 2.

The X-axis is the length of bit stream N and the Y-axis depicts the total number of viable parsing schemes of the test bit stream which is expressed in the log scale with base 2. Several specific values of $V(N)$ are given as follows: $V(4) = 3$, $V(13) = 46$, $V(18) = 337$. In particular, $V(24) = 2084$. The figure clearly shows that the values of $V(N)$, $1 \leq N \leq 24$, form a straight line with a certain slope, indicating an exponential rate of growth of $V(N)$ with respect to bit stream length N.

In summary, the above study indicates that there exist multiple viable parsing schemes for a general RHT-encrypted bit stream. Furthermore, the size of viable parsing schemes, $V(N)$, grows exponentially with bit stream length, N. Thus, the parsing attack does not provide a significant advantage over the ciphertext-only attack, in which the search space is also an exponential function of N. Hence, RHT encryption can withstand the parsing attack.

2.6.3 Known-Plaintext and Chosen-Plaintext Attacks

In the known-plaintext attack, the cryptanalyst has access to several plaintext/ciphertext pairs encrypted with the same key. In the case of chosen-plaintext attack, the cryptanalyst is allowed the additional freedom to choose the plaintext at his/her own will and observe the corresponding ciphertext. They are more powerful attacks than the ciphertext-only attack.

The security of REC-based schemes under these two types of attacks is closely tied with the Alias Coding Parameters (ACP) in Definition 2. Due to the existence of ACP, there exist multiple KHS values for a given plaintext/ciphertext pair and they are called the *alias KHS*. This prevents a cryptanalyst from determining a unique KHS value by observing plaintext/ciphertext pairs. This is stated in the following theorem.

Theorem 2 *Under the known-plaintext and the chosen-plaintext attacks, for a given plaintext* $p = p_0\, p_1 \ldots p_L$, *the total number of alias KHS is given by*

$$\prod_{0 \le i \le L} A(p_i)$$

where $A(p_i)$ is the alias degree of each plaintext symbol p_i.

Proof: Suppose that the KHS corresponding to plaintext $p = p_0 p_1 \ldots p_L$ is $z = z_0 z_1 \ldots z_L$. The goal of the cryptanalyst is to recover sequence z. Note that REC-decryption is an L-step progressive operation. In step i, the cryptanalyst decrypts the ciphertext using all ECP values and check the decryption result with p_i to determine the value of z_i. The process continues for the remaining ciphertext until all plaintext $p_i's$ have been decoded. By definition of the alias degree, the number of ECP values in step i that gives the correct plaintext p_i is $A(p_i)$. For example, in the very first step, a cryptanalyst finds that there are $A(p_0)$ ECP values that would lead the entropy coder to decode an output of p_0. In the second step, $A(p_1)$ ECP

values give an output of p_1, and so on. Thus, each component KHS z_i has $A(p_i)$ possible values corresponding to the $A(p_i)$ ECP values that yield the same plaintext output. The total number of alias KHS is equal to the product of all $A(p_i)$; namely, $\prod_{0 \le i \le L} A(p_i)$. This completes the proof. ■

The safeguard against the known-plaintext and the chosen-plaintext attacks relies on the alias degree of plaintext symbols. The larger the alias degree, the higher the security against these attacks. The alias degree is related to the underlying entropy coder, the construction of the ECP pool and the design of encryption algorithms. In the design of RDT encryption scheme, we intentionally split all dictionary tables into two halves and insert the new entry to the same (random) position in each half to maximize the alias degree.

It should be noted that the above analysis of the alias degree is plaintext-dependent. If several plaintext/ciphertext pairs associated with the same KHS are available, the cryptanalyst might narrow down the number of alias keys or even determine some part of the KHS uniquely. In the chosen-plaintext attack, it could be possible to recover the entire KHS by selecting the input plaintext carefully.

2.7 Comparsion of REC and Cryptographic Ciphers

The REC-based encryption schemes and traditional cryptographic ciphers are compared in this section. Here, it is assumed that the pseudo-random sequence is generated by a stream cipher since it has a low computational cost and does not affect the security strength.

2.7.1 Comparison with Block Cipher

The encryption speed of REC-based schemes is faster than that of a block cipher since the pseudo-random sequence is generated by a stream cipher. In terms of cryptographic

strength, the REC model is at least as strong as the underlying stream cipher with respect to the ciphertext-only attack. So far, there is no absolute conclusion about the security comparison between the block cipher and the stream cipher. However, considering the fact that these are two main types of ciphers used in the security world, their security strength can be viewed as comparable. The resistance of REC-based schemes to the known-plaintext and the chosen-plaintext attacks relies on the existence of multiple alias KHS. As discussed above, a cryptanalyst can narrow down or even reveal the KHS when an enough number of plaintext/ciphertext pairs are accumulated or using carefully chosen plaintexts while a block cipher can usually withstand these attacks with fairly good strength. Thus, the security of REC-based schemes under the known-plaintext and the chosen-plaintext attacks is weaker than that of a block cipher.

2.7.2 Comparison with Stream Cipher

REC-based schemes are roughly as fast as a stream cipher. Other than the common cost of generating a pseudo-random keystream, a stream cipher takes an additional XOR instruction to perform encryption. The extra cost of REC-based schemes is to let the entropy coding take multiple ECP values, which is small. Both give comparable performance in terms of the encryption speed. The main advantage of the REC-based approach lies in the security strength as compared to the stream cipher. In a stream cipher the ciphertext is simply the bit-wise XOR of the plaintext and the keystream so that a single plaintext/ciphertext pair enables one to recover the keystream (or KHS in the REC approach). For REC-based schemes, this is not the case due to multiple alias KHS. If the number of plaintext/ciphertext pairs available to a cryptanalyst is limited, the REC approach provides a better security protection than a stream cipher against the known-plaintext and the chosen-plaintext attacks.

2.8 Experimental Results

To demonstrate the advantages of the proposed REC encryption schemes, in this section we implement the RHT encryption scheme in the H.264 video compression and conduct some experiments. First, we show the visual effect of the encrypted video data. The compressed video bit stream is also encrypted using conventional block cipher and stream cipher. The computation efficiency of RHT encryption and cryptographic cipher are compared. All simulation programs are coded in C language and run on a Pentium-M 1.4 GHz processor with 512MB RAM space.

2.8.1 Visual effects of RHT encryption

We implemented the RHT encryption algorithm and embedded it in the H.264 video compression system. The 128-bit MD5 hash algorithm was used to generate the KHS. The test video clip used was the "Foreman" sequence of the CIF size. The first 10 frames of the "Foreman" sequence were encrypted with the RHT scheme with the key "0x246CCA6B103C95" (in hexadecimal representation). The following two attacks are performed.

Attack 1. At the decoder side, the encryptedbit-stream was decoded normally *without* the RHT decryption algorithm. In this test, we would like to demonstrate the effect of decoding an RHT-encrypted bit-stream using the standard H.264 decoding procedure.

Attack 2. At the decoder side, the encrypted bit-stream was decoded using the RHT decryption algorithm but with some random keys. One was a randomly selected number "0x17460FD05B9EDF" while the other is chosen to be "0x246CCA6B103C94", which differs from the correct key by only a bit. The test of the first key simulates the scenario where an adversary attempts to attack the proposed RHT scheme by decrypting the bit-stream using

a randomly selected key. The test of the second key is to see the performance of two keys which are close in the key space.

The original 1st and 9th frames of the Foreman sequence are shown in Fig. 2.6. The decoded results for these two frames are given in Figs. 2.7-2.9 using no key, the key of "0x17460FD05B9EDF" and the key of "0x246CCA6B103C94". From these figures we can see that the decrypted images in all attacks are totally meaningless. This indicates that RHT algorithm provides satisfactory encryption performance which is suitable for video content protection.

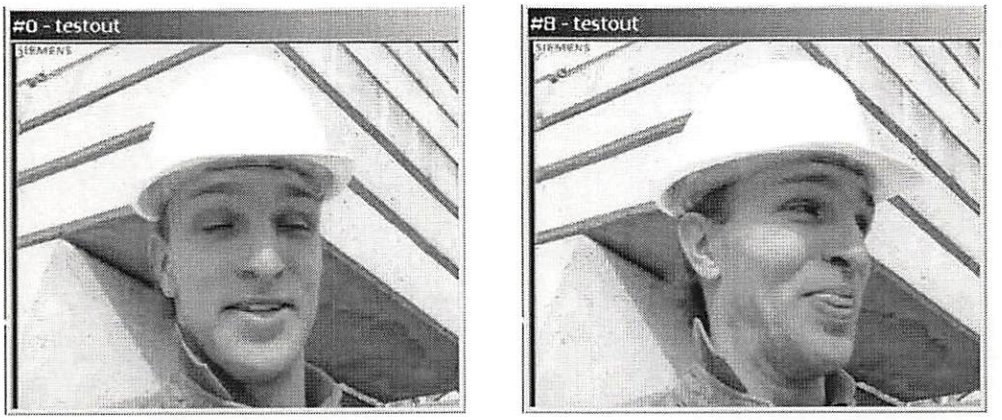


Figure 2.6: (a) Frame no. 1 and (b) frame no. 9 of the "Foreman" video clip

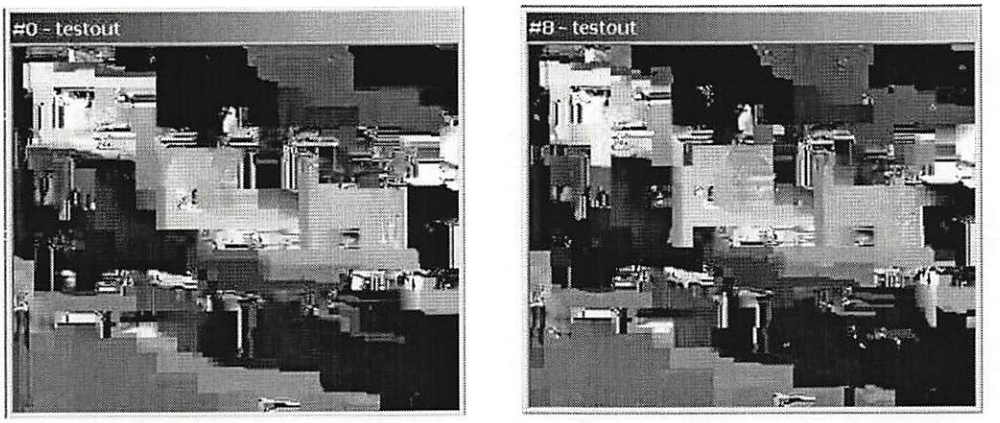


Figure 2.7: (a) Frame no. 1 and (b) frame no. 9 of the RHT-encrypted "Foreman" video clip decoded using the standard H.264 decoder

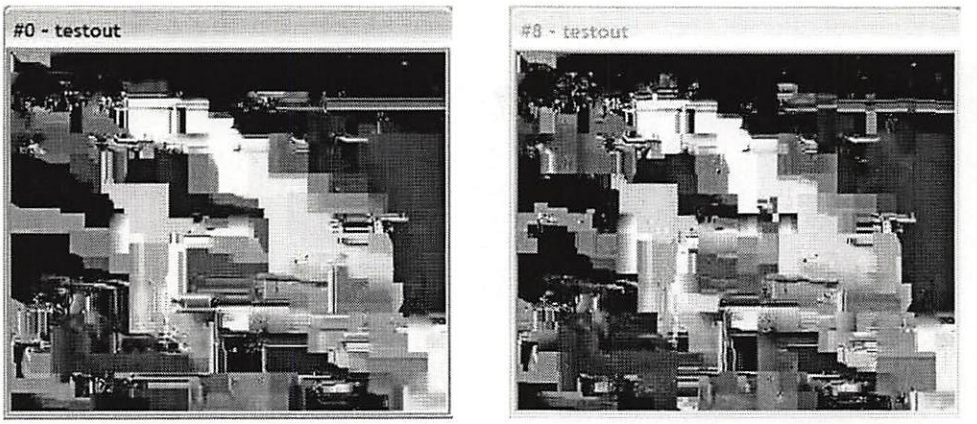


Figure 2.8: (a) Frame no. 1 and (b) frame no. 9 of the "Foreman" video clip decoded using the RHT algorithm with key 0x17460FD05B9EDF

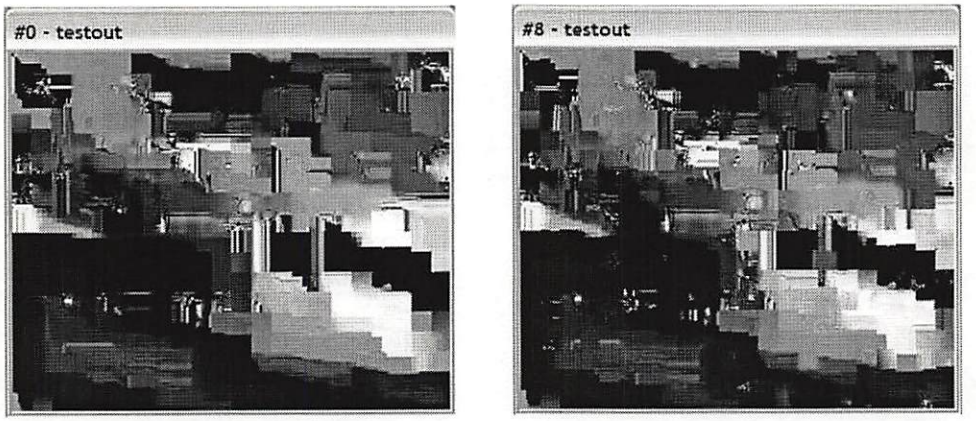


Figure 2.9: (a) Frame no. 1 and (b) frame no. 9 of the "Foreman" video clip decoded using the RHT algorithm with key 0x246CCA6B103C94, which differs from the correct key by only 1 bit

2.8.2 Computation efficiency comparison with cryptographic ciphers

To examine the computational efficiency of the RHT scheme, we encrypt the compressed bit stream file using conventional cryptographic ciphers and compare their encryption time with that of the RHT scheme.

In our experiment, the DES and the RC4 encryption algorithms are used as a representative of the block cipher and the stream cipher, respectively. For fair comparison, RC4 is also used as the KHS generator in RHT encryption. To alleviate accidental variations in running time, we encode and encrypt the first 100 frames of the "Foreman" video using RHT, DES, and RC4, respectively. Since RHT encryption is easily embedded into the entropy coding, its running time is measured as the total time of encoding with RHT encryption minus the total time of encoding only. For each cipher, the same experiment is executed 10 times and the run time statistics is calculated as the average. The results are shown in the following table. The unit of time is millisecond. Notations in the table are explained below:

T_{code}: time of encoding the 100 frames only

T_{total}: time of encoding plus encryption

$T_{enc} = T_{total} - T_{code}$: time spent on encryption

$T_{enc}/100$: time to encrypt one frame

T_{enc}/T_{total}: the percentage of time spent on encryption

	T_{code} (ms)	T_{total} (ms)	T_{enc} (ms)	$T_{enc}/100$ (ms)	T_{enc}/T_{total}
RHT	13328	13506	178	1.78	1.34%
RC4	13328	13491	163	1.63	1.22%
DES	13328	13669	341	3.41	2.56%

Table 2.1: Encryption time comparison of RHT, DES and RC4.

We see from this table that RHT encryption is almost as fast as the RC4 stream cipher. This is due to the fact that the dominant computation cost of RHT encryption is the generation of KHS, while selection of different Huffman tables is done by a simple array index operation whose cost is negligible. Both RHT encryption and stream cipher RC4 are much faster than block cipher DES. In summary, the proposed RHT encryption scheme demands a very low computation cost. It can achieve an encryption speed, which is the same as a stream cipher and much faster than a block cipher.

2.9 Conclusion

A new encryption approach called the Randomized Entropy Coding (REC) is proposed in this research. The basic idea is to apply multiple entropy coding parameters (settings) dynamically according to a statistically random sequence to encode consecutive inputs. Based on the REC approach, three specific encryption schemes were developed using the Huffman, the arithmetic, and the LZ78 coders as the underlying entropy coders, respectively. The REC encryption approach only involves the selection of coding parameters in a random manner. Thus, it has a low computational cost and can be easily integrated into a multimedia compression system by either software or hardware. It does not impair the coding gain of the original data compression scheme. The REC method was proved to be secure under the ciphertext-only attack. Its resistance to known/chosen-plaintext attack is good due to the existence of multiple alias keys, which encrypt a given plaintext to the same ciphertext.

Chapter 3

Multimedia Encryption via Coded Bit Stream Shuffling

3.1 Introduction

Although encrypting the entire bit stream by a traditional cryptographic cipher (*e.g.*, the block or stream cipher) achieves a satisfactory level of security, the complexity is nevertheless high. In emerging mobile wireless multimedia applications, portable devices such as cellular phones or hand-held consumer electronics are heavily constrained in resources due to the limitation in the size and power consumption. They are designed to support only a certain set of basic functions, and it is difficult for them to accommodate the heavy computing of total encryption. Thus, it is desirable to develop an efficient yet secure bit stream encryption technique.

In this chapter, the problem of light-weight multimedia encryption is examined from a new angle. That is, regardless of the underlying compression techniques, the compressed bit stream can be treated as a nearly random binary sequence. By exploiting the randomness of plaintext data, we propose a light-weight encryption algorithm called the "Random Rotation in Partitioned Blocks (RPB)" scheme. The basic idea is to partition the data into random-sized blocks and perform a random rotation in each block according to a pseudo-random

keystream. The resultant encryption scheme demands low computational complexity because of the simplicity of partition and rotation operations. Analysis shows that the key space of a brute-force attack is an exponential function of the plaintext/ciphertext length. The RPB scheme is thus secure under the ciphertext-only attack since a brute-force attack seems to be the only effective strategy given the randomness nature of the plaintext. An important property of the RPB encryption is observed that there exist multiple keys that encipher the same plaintext to the same ciphertext. Such keys are defined as alias keys. We demonstrate that the average number of alias keys grows exponentially with the size of the plaintext/ciphertext. The RPB scheme can therefore resist the known/chosen-plaintext attack due to the large alias key space. Using a stream cipher as the keystream generator, the PRB encryption scheme basically achieves the block cipher's security strength at the stream cipher's encryption cost. We also study the application of the proposed RPB scheme to multimedia content encryption. To justify the random plaintext requirement, experiments are conducted to evaluate the statistical properties of the multimedia bit stream.

The rest of this chapter is organized as follows. In Sec. 3.2, we discuss the relationship between plaintext redundancy and encryption complexity and propose a methodology of cascading a randomization unit after compression to achieve efficient encryption. The general idea and the detailed construction of the RPB encryption scheme are presented in Sec. 3.3. The computational cost is analyzed in Sec. 3.4. The key space and the alias key property of the RPB scheme are studied, and the security performance with respect to the ciphertext-only attack and the known/chosen plaintext attack is studied in Sec. 3.5. The statistical randomness relationship between the input and the output is investigated based on the entropy change analysis in Sec. 3.6. Comparison of the proposed RPB scheme with the block cipher and the stream cipher is made in Sec. 3.7. The application of the proposed

RPB scheme to multimedia data encryption is conducted in Sec. 3.8. Finally, concluding remarks are given in Sec. 3.9.

3.2 Compression-based Multimedia Data Encryption

In essence, encryption is a process of transforming a plaintext that has some structure and semantics to a random ciphertext that has no observable structure. The ciphertext is related to the plaintext under the control of a secret key. During the cryption process, the structure and semantics of the plaintext is removed via processing so that the plaintext is converted to a random-looking ciphertext. Generally speaking, the plaintext contains a significant amount of redundancy among symbols/bits while the bitstream of the ciphertext is almost free from redundancy.

Thus, from the information theory viewpoint, encryption can be regarded as a redundancy-removing transformation. To achieve this goal, traditional cryptographic ciphers use heavy-weight operations and bit manipulations to scramble the plaintext. The block cipher and the stream cipher are two basic types of cryptographic ciphers. In a block cipher, data are divided into blocks of certain length, and these blocks undergo several rounds of bit-manipulating operations, including substitution, arithmetic operation, permutation, linear affine transformation, etc., to produce the ciphertext. A stream cipher, on the other hand, enciphers by XORing the plaintext with a keystream that is the output of a pseudo-random bit generator.

It appears that there lies a tradeoff between the plaintext redundancy, the complexity of the encryption operation, and the security strength of the encryption algorithm. Generally speaking, the higher the redundancy a plaintext contains, the more complex the encryption operations are needed to achieve a certain level of security. To put it in another way, for a given plaintext, a more complex encryption process should lead to higher security strength.

This observation can be justified by some well known facts. Take the DES (Data Encryption System) as an example. DES is a block cipher with 16 rounds of operation. It was discovered recently [20] that DES with less than 16 rounds is vulnerable to certain linear and differential attacks.

The redundancy of plaintexts does not receive adequate attention in previous cryptographic research, since the security strength of the encrypted data is much more important than the efficiency of the encryption process. Traditionally, the design goal of a cipher has focusd more on the cipher's ability to withstand known attacks for all possible plaintexts, regardless of its redundancy characteristics. It has been considered rather useless to design a specialized cipher for a particular type of plaintexts with certain characteristics and redundancy level.

However, the situation changes when it comes to multimedia data encryption. Multimedia data (mostly, video and audio contents) rarely exist in raw format. Instead, they are usually compressed to a bit stream file before they are transmitted, exchanged and stored. As a result of effective coding algorithms, most redundancy in the raw data has been removed and the output bit stream can be considered a nearly-random, almost redundancy-free binary sequence.

Since the compressed multimedia bit stream provides a special type of low-redundancy plaintexts, it could be possible to design a light-weight encryption scheme to achieve a security level comparable with that of the traditional cryptographic cipher based on the use of high complexity operations. This is the research motivation of our current work. Due to the wide availability of compressed multimedia data, such an efficient and secure multimedia encryption scheme is of great practical value.

To design a light-weight encryption scheme dedicated to coded multimedia data, we first examine the relationship between compression and encryption. As mentioned above, compression provides a redundancy-removing transformation that converts large-size, highly-redundant raw contents into smalle-size, almost redundancy-free contents. We compare the input and the output of encryption and compression in Table 3.1.

Encryption	Size	Redundancy	Randomness	Decryption
Input	same	high	not random	
Output	same	low	random	***requires secret key***
Compression	**Size**	**Redundancy**	**Randomness**	**Decompression**
Input	large	high	not random	
Output	small	low	***nearly random***	no key required

Table 3.1: Comparison of encryption and compression.

We see from this table that encryption and compression do share some similarities. There are however two important distinctions between them, which are highlighted by the slanted boldface font in the table. First, the size of the output file is reduced to a small percentage of that of the input file in compression while the size of the output file is exactly the same as that of the input file and there exists no length reduction in encryption. Second, a secret key is required to decrypt an encrypted ciphertext into the original plaintext while no secret key is needed to decode the compressed bit stream to obtain the raw multimedia content in the decompression process.

Given the above observations, one can view compression as a special type of encryption since it removes the redundancy in the input file to yield a nearly random output file. If the compression algorithm is not known to the attackers, it would be very difficult to recover the original bit stream. In many practical applications, the compression/decompression algorithms are open to the public for the compatiblity consideration. In other words, we cannot hide the algorithm in order to protect the content. However, we may turn an source encoder

into a secure cipher by introducing a secret key and a light-weight encryption module to prevent the recovery of the input file from the output file as done in a regular decompression process. It is possible to use a light-weight encryption module in this system, since compression has already done a large amount of job needed to encrypt the input, *i.e.* to remove the redundancy of the input data.

In this work, we propose a specific scheme to realize the light-weight encryption module. That is, we place a simple randomization unit directly after the source encoder. With such a randomization operation, we can scramble the coded bit stream under the control of a secret key. The proposed "*compression cascade encryption*" system is shown in Fig. 3.1.

In this system, M is the highly-redundant multimedia content in raw format. After the compression system, it is encoded to a bit stream file A, whose size is much smaller than M due to redundancy removal. Then, the randomization unit transforms A to ciphertext stream C by incorporating a randomization process controlled by a key. It is computationally efficient and structurally simple as compared to traditional cryptographic ciphers. The light-weight nature allows the implementation of the randomization unit as an integral part of a compression system in a VLSI chip. Owing to this reason, we treat the compression system cascaded with the light-weight randomization unit as one inseparable box, which is enclosed by the dashed line in the figure. All attacks are performed with respect to the input and output relationship of the dashed-line enclosed box. In the next section, we propose a simple way to implement the randomization unit.

3.3 Random Rotation in Partitioned Blocks (RPB)

Throughout this work we treat the bit stream A and C as a 0-1 binary sequence. To design a randomization unit with light-weight operations, our idea is to exploit the order of bits in

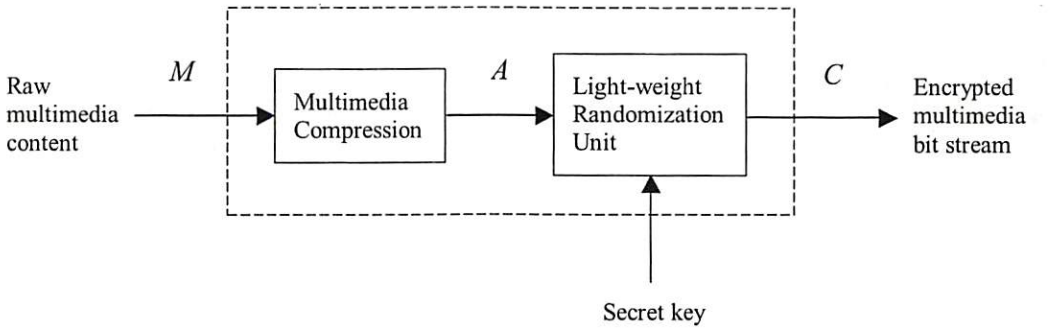


Figure 3.1: The block diagram of a cascaded compression-encryption system.

the bit stream. Here, we propose two techniques to encrypt the bit stream. First, the 0-1 bits are grouped into blocks of a random size. Second, we alter the order of individual bits in each of these blocks. Both the length of bit blocks and the change of order are selected according to a random sequence, which is the encryption key sequence.

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 bit stream processing, we restrict the bit manipulation to a simple *left rotation* here. For a block of n bits $A = (a_1 a_2 \dots a_n)$, an r-bit left rotation transforms A into $(a_{r+1} a_{r+2} \dots a_n a_1 a_2 \dots a_r)$ 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 bit stream for the final output, thus adding a very small computation overhead. Furthermore, although left rotation is a simple operation, our analysis in Sec. 3.5 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 4 *Let $A = (a_1 a_2 \dots a_N)$ be a bit stream of length N. The (p, r) rotation in partitioned blocks of A, denoted $RPB(A, p, r)$ with $p = (p_1 p_2 \dots p_m)$ and $r = (r_1 r_2 \dots r_m)$, 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 Fig. 3.2 to illustrate the RPB operation applied to a stream of 10 bits $A = (a_0 a_1 \ldots a_9)$. The partition sequence is $p = (3, 5, 2)$ and the rotation sequence is $r = (2, 3, 1)$. The bit stream after performing $RPB(A, p, r)$ is denoted by C.

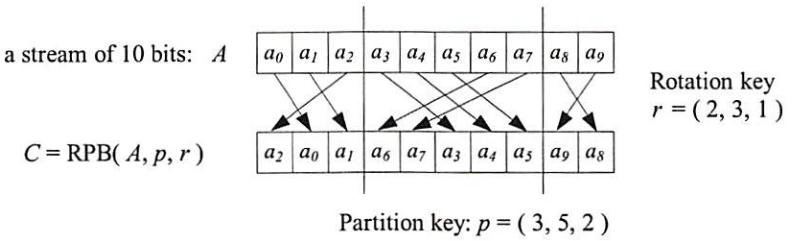


Figure 3.2: An example of rotation in partitioned blocks

In the proposed RPB encryption scheme, a plaintext bit stream A is enciphered into a ciphertext $RPB(A, p, r)$ 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 pseudo-random bit sequence, which is generated by a PRBG using a secret seed.

The RPB algorithm has another benefit when the plaintext to be encrypted is output of data compression system. In such system bits are continuously generated from certain entropy coder to form the final compressed bit stream. This allows the partition and rotation operation be conveniently done by the following procedure. For a single p-bit block A, an r-bit left rotation is equivalent to a "delayed-write" as conducted in below steps:

1. Hold the first r bits of A;

2. Put the remaining $p - r$ bits to the buffer;

3. Put the r bits in Step 1. to the buffer.

The above "delayed-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 the delayed-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.

Random Rotation in Partitioned Block (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 $C = RPB(A, p, r)$ 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 $RPB(C, p, p - r)$ recovers the plaintext A from the ciphertext C.

In real-world data compression systems, the bit stream is generated by the entropy coder in a sequential order. To avoid frequent memory write, these bits are not written to the final bit stream file directly. Instead, they are first stored in a buffer at the end of the entropy coder. When an enough number of bits are collected, the entire buffer content is written into the bit stream file by a single memory write operation.

3.4 Computational Cost Analysis

The RPB encryption scheme is in essence a bit reordering algorithm in variable-length blocks of the plaintext bit stream. 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 "delayed-write" operation described in the last section.

In practice, it is quite very to implement the "delayed-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 pseudo-random sequences to yield the partition sequence and the rotation sequence.

3.5 Security Analysis

We study the security of the RPB unit from two angles. First, we introduce the alias key concept, and analyze the key space as well as the alias key space associated with the RPB. Attack complexity under the ciphertext-only attack and the known/chosen-plaintext attack

can be evaluated accordingly. Second, we examine the randomness of the input and the output streams from an information theory viewpoint. That is, we prove that there is no entropy change between the input and the output streams.

3.5.1 Key Space Analysis

The security of the RPB encryption scheme relies on the size of the key space. For a given N-bit ciphertext $C = RPB(A, p, r)$, 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 $C = RPB(A, p, r)$, then the plaintext is $A = RPB(C, p, p - r)$. 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 5 *Let $A = (a_1 a_2 \ldots a_N)$ be a bit stream of length N. Two RPBs of A, $RPB(A, p_1, r_1)$ and $RPB(A, p_2, r_2)$, 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 $A = (a_1 a_2 \ldots a_N)$ 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 2 *Let $A = (a_1 a_2 \dots a_N)$ be a bit stream 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:*

$$R(N) = 2R(N-1) + \sum_{k=N-B}^{N-3} R(k) \tag{3.1}$$

and

$$R(N) > 2^N, \quad \textit{for} \quad N \geq 6 \tag{3.2}$$

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 Eq. (3.1). From this recursive equation, the lower bound given in (3.2) is straightforward since $R(N) > 2R(N-1)$. A detailed proof is provided in Appendix A.2.

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.

3.5.2 Alias 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 alias key, and show how it can help defend known/chosen-plaintext attack.

In Definition 5, 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 bit stream, 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:

8-bit plaintext: $A = (a_1 a_2 \dots a_8)$

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, $RPB(A, p_1, r_1) = (a_1 a_3 a_4 a_5 a_6 a_7 a_8 a_2)$ and $RPB(A, p_2, r_2) = (a_3 a_1 a_2 a_5 a_6 a_7 a_4 a_8)$. They are apparently different RPBs by Definition 5. However, for a particular plaintext $A = (01011101)$, it is readily checked that $RPB(A, p_1, r_1) = RPB(A, p_2, r_2) = (00111011)$. That is, both keys encipher A to the same ciphertext $C = (00111011)$. These keys are called *alias keys*. Mathematically, the alias key is defined as follows.

Definition 6 (Alias keys) *For a given plaintext bit stream A, two keys (p_1, r_1) and (p_2, r_2) are called alias keys, if*

1. *$RPB(A, p_1, r_1)$ and $RPB(A, p_2, r_2)$ are different RPB per Definition 5.*

2. *They transform A to the same output stream $C = RPB(A, p_1, r_1) = RPB(A, p_2, r_2)$.*

We stress that the concept of alias keys is associated with a particular ciphertext (assuming a fixed plaintext). Two alias keys for one ciphertext may not be alias keys for another ciphertext. Discussion on alias 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 alias keys. First, does there exist alias keys? Second, if there is any then what is the exact amount of alias 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 alias keys by observing the bit stream pattern and do several trials. The second problem, *i.e.*, to compute the accurate number of alias keys, is however not an easy one. Since alias keys are ciphertext-dependent, there seems no quick formula to compute the number of alias 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 alias keys.

Lemma 3 *Let $A = (a_1 a_2 \dots a_N)$ be a bit stream of length N containing Z 0's and $Alias(A, C)$ denote the number of alias keys for the plaintext/ciphertext pair (A, C). Then, there exists a ciphertext A' such that*

$$Alias(A, A') > \left\lceil 2^N \Big/ \binom{N}{Z} \right\rceil \tag{3.3}$$

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

$$Alias(A, A') > \sqrt{\pi N/2} \tag{3.4}$$

The above lemma establishes the existence of alias keys. Refer to Appendix A.3 for a complete proof. The quantity $\sqrt{\pi N/2}$ is however a conservative estimate of number of alias keys. Further analysis of the average number of alias keys will be given in Sec. 3.5.5 below.

3.5.3 Ciphertext-Only Attack

To evaluate the security strength of the proposed RPB encryption scheme, we consider ciphertext-only and known/chosen-plaintext attacks in this and the next subsections.

Given the randomness nature of plaintext data, a cryptanalyst would have to resort to an exhaustive search of the keystream in a ciphertext-only attack. Thus, breaking the RPB encryption is at least as hard as the underlying PRBG algorithm. If an algorithm can successfully decrypt an RPB-encrypted ciphertext, such an algorithm can be used to produce a pseudo-random sequence without knowing the seed, thus breaking the underlying PRBG algorithm.

In a brute-force attack to find out the partition sequence and the rotation sequence, a cryptanalyst can either exhaust all $R(N)$ possible RPBs or try all r-bit seed of the PRBG algorithm. The computation complexity is $\min(2^r, R(N))$. As shown by lemma 2, $R(N) > 2^N$. The size of $R(N)$ increases exponentially with the length of ciphertext. This is a very important property of the RPB operation that would thwart any brute-force effort if the bit stream is long enough. For instance, in the state-of-the-art video compression standard such as H.264, it would cost no less than $1 \sim 2$ kilo-bits to encode a CIF-size (352x288) video frame. Given that $N >> r$ in practice, the cryptanalyst would rather attack the underlying PRBG algorithm. The RPB scheme remains secure under ciphertext-only attack if a secure PRBG algorithm is used as the keystream generator.

3.5.4 Chosen-Plaintext Attack

Under the chosen-plaintext attack, a cryptanalyst can select any plaintext of his/her choice and observe the corresponding ciphertext. This is the most favorable case for the cryptanalyst and the biggest threat to a cryptographic cipher.

The security of the RPB encryption scheme highly depends on the plaintext in the system diagram in Fig. 3.1. If bit stream A, which is the plaintext to the RPB encryption module, can be chosen freely by a cryptanalyst, RPB encryption succumbs to the chosen-plaintext

attack since the encryption is a simple bit-reordering scheme. By applying input streams of particular patterns, it is not difficult to determine partition sequence p and rotation sequence r. One such attack algorithm was proposed by Chia-Mu Yu at the Institute of Information Science, Academia Sinica, Taiwan. This algorithm requires $O(N)$ chosen plaintexts and $O(N)$ time complexity. Thus, the RPB encryption cannot withstand the chosen plaintext attack when it is used as a stand-alone module. However, as emphasized in Sec. 3.2, the multimedia coding system and the RPB unit should be treated as an inseparable black box and the coded bit stream A in Fig. 3.1 is hardly accessible by attackers. In other words, the cryptanalyst cannot launch a chosen-plaintext attack on the RPB encryption unit.

On the other hand, a cryptanalyst is allowd to choose the raw multimedia content M as the input to the black box. Since high performance multimedia coding algorithms eliminate the redundancy in the input multimedia file, the output bit stream A will appear as a random bit stream as the input to the RPB encryption module, which is usually called the known-plaintext attack. The security of the RPB encryption module under the known plaintext attack will be examined in the next subsection.

3.5.5 Known Plaintext Attack

Under the known-plaintext attack, several plaintext/ciphertext pairs are available to study. As discussed above, those plaintext/ciphertext pairs are general random samples without special characteristics. In this case, for a given plaintext, there exist multiple alias keys that encipher it to the same ciphertext. Thus, the cryptanalyst cannot differentiate these alias keys given only a few plaintext/ciphertext pairs. To uniquely determine the correct key, a cryptanalyst needs a lot of plaintext/ciphertext pairs encrypted using the same key so as to rule out wrong keys. Of course, the larger the number of alias keys for a general ciphertext,

the more pairs are needed and hence the greater the cryptanalyst's effort to find out the correct key. The resistance to the known-plaintext attack thus counts heavily on the number and characteristics of alias keys for a general pair of plaintext/ciphertext.

It is shown in lemma 3 that the number of alias keys $Alias(A, C)$ is larger than $\sqrt{\pi N/2}$ for at least one ciphertext A'. 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 alias keys of different ciphertexts do not overlap. In fact, we can show by plausible reasoning that the number of alias keys far exceeds $\sqrt{\pi N/2}$. It also grows exponentially with respect to the ciphertext length N.

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

Lemma 4 *$A(N) \sim c^N$ 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 $A(N)$ alias keys for a general plaintext A_1 and k_2 is any one of the $A(N)$ alias keys for another plaintext A_2 then a concatenation key $k = k_1||k_2$ is alias key for the plaintext concatenation $A_1||A_2$. A function $A(N)$ satisfying this property must be of the form c^N. Appendix A.4 gives a detailed proof.

We emphasize that c^N is not an accurate formula of $A(N)$ but it depicts the asymptotic behavior of $A(N)$ with respect to N. For a sufficiently large value of N (a long enough plaintext), the average number of alias keys for a general plaintext quickly becomes intractable since it is exponential with the plaintext length. This exponential growth rate of the number

of alias keys, $A(N)$, plays a key role in RPB encryption's ability to withstand the known-plaintext attack.

Due to the large amount of alias keys, a cryptanalyst 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 alias 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, a cryptanalyst can proceed as follows. At first, he/she can select a given pair and calculate $A(N)$ alias keys for this pair using a certain algorithm. Then, he can check all these $A(N)$ keys against each available pair (A_i, C_i). If $RPB(A_i, k) = C_i$, then k is counted as one possible key. Otherwise, k must be a wrong alias 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 5 *With $R(N)$, $A(N)$ and $P(N)$ defined in the above description, we have the following relationship:*

$$P(N) = \frac{R(N)}{R(N) - A(N)} \ln R(N). \tag{3.5}$$

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

$$P(N) \approx N \ln 2. \tag{3.6}$$

To solve this problem, we may examine it from another angle. That is, X containing only one key is equivalent to saying that $\bar{X}$, the complementary set of X, contains $R(N)-1$ keys. By treating $\bar{X}$, we can convert this problem to a variant of the classical coupon collector problem. Please refer to Appendix A.5 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 an attack is still formidable due to the exponential growth rate of alias key number $A(N)$ with plaintext length N. It was mentioned before that in the black box structure, a cryptanalyst is only allowed to manipulate raw data M but prohibited from directly manipulating A, which is the input to the RPB unit. In other words, it is difficult for a cryptanalyst to obtain input 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 compression-RPB scheme has strong resistance against the known-plaintext attack.

3.5.6 Numerical Evaluation of A(N)

As demonstrated above, the resistance of RPB encryption scheme to known-plaintext attack is attributed to the existence of multiple alias keys that encipher a given plaintext to the same ciphertext. We have proved that the average number of alias keys for a general N-bit stream also grows exponentially with N, that is, $A(N) \sim c^N$ for $c > 1$, but did not give the exact value or a range of c. Table 3.2 provides a rough idea of how large $A(N)$ is for a different pair of c and N. The size of N is typically of order 10^4 in bits. From the table, we see that $1.01^{10000} \approx 1.636 \times 10^{43} \approx 2^{143}$. Thus, even if c is very small, in practice the size of $A(N)$ is already impossible to attack with commonly available computing power.

$c\backslash N$	100	200	500	1000	2000	5000	10000
1.01	2.705	7.316	144.773	20959	4.393×10^{8}	4.044×10^{21}	1.636×10^{43}
1.02	7.245	52.485	19957	3.983×10^{8}	1.586×10^{17}	1.002×10^{43}	1.004×10^{86}
1.05	131.501	17293	3.932×10^{10}	1.546×10^{21}	2.391×10^{42}	8.841×10^{105}	7.816×10^{211}
1.1	13780	1.899×10^{8}	4.970×10^{20}	2.470×10^{41}	6.101×10^{82}	9.192×10^{206}	8.450×10^{413}
1.2	8.282×10^{7}	6.859×10^{15}	3.896×10^{39}	1.518×10^{79}	2.304×10^{158}	8.058×10^{395}	6.493×10^{791}

Table 3.2: Size of $A(N) = c^N$ with respect to c and N

3.6 RPB's Impact on Statistical Randomness of A and C

In previous sections, we study the security of the RPB scheme under several attack modes from a cryptographic viewpoint. In this section, the security of the RPB scheme is examined from another angle. We investigate the statistical behavior of the RPB unit by analyzing its impact on the statistical randomness of input stream A and output stream C. It is apparent that the light-weight randomization unit should not weaken the statistical randomness of its input stream A. In other words, the output stream 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 streams A and C. This is stated in the following lemma.

Lemma 6 *Let A be a general input bit stream of length N and C be the output bit stream of the RPB encryption unit. Then, for a sufficiently large value of N, the first, second and third-order entropies of A and C are equal,* i.e.,

$$\begin{aligned} H^{(1)}(A) &= H^{(1)}(C), \\ H^{(2)}(A) &= H^{(2)}(C), \\ H^{(3)}(A) &= H^{(3)}(C). \end{aligned} \tag{3.7}$$

It is straightforward to show $H^{(1)}(A) = H^{(1)}(C)$ 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.6.

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 entropy 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 $H(A) = H(C)$ 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.

3.7 Comparison with Cryptographic Ciphers

Assume that a stream cipher based PRBG generator is used in the RPB encryption scheme to produce the random partition sequence and the rotation sequence. As compared to the block cipher, the RPB scheme achieves a faster encryption speed than that of the block cipher. The dominating computation cost of the proposed RPB scheme is the keystream generation by stream cipher, which is much faster than the block cipher. As far as security strength is concerned, the RPB scheme far exceeds a stream cipher. That is, for a sufficiently long plaintext/ciphertext of size N, the RPB scheme can withstand both the ciphertext-only attack and the known/chosen-plaintext attack by a complexity of $R(N)$ and $A(N)$, respectively, as

shown in our analysis. Thus, the security level of the RPB scheme is exponential with the ciphertext size N, which is comparable with that of the block cipher. In contrast, a single plaintext/ciphertext pair would reveal the keystream for a stream cipher.

3.8 Case Study: Multimedia Data Encryption

In this section, we examine multimedia data encryption as one application area of the proposed RPB scheme. Since a compressed multimedia bit stream contains little redundancy and can be viewed as nearly random, the RPB scheme is an appropriate choice to secure multimedia content. Previous multimedia encryption schemes are first reviewed. Properties of compressed multimedia bit stream are discussed and statistical experiments are conducted to verify the randomness of such data. 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.

3.8.1 Review of Bit-Stream-Oriented Encryption Schemes

A general approach to secure multimedia content is to encrypt the resulting bit stream from the compression unit. To encrypt the entire bit stream using traditional cryptographic ciphers such as block cipher imposes a heavy computational burden on both encryption and decryption units due to the large size of the bit stream. Thus, several low-complexity encryption schemes have been proposed. They are reviewed in this section.

An encryption scheme called "Aegis" was developed by Spanos and Maples [5], which encrypts only the I-frames of a MPEG video stream. Intuitively, B and P frames cannot be correctly decoded without the help of the I frames. However, Agi and Gong [6] showed that a considerable amount of video content is still visible due to the unencrypted I-macroblocks in B and P frames as well as the interframe correlation. Another scheme called SECMPEG was

developed by Meyer and Gadegast [21], which 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 the use of additional headers.

Qiao and Nahrstedt [11] proposed a scheme to split a bit stream into two halves. They are called the *odd* list and the *even* list according to a random pattern. The ciphertext c is obtained by the following operation

$$c = odd \oplus even$$

where $\oplus$ is the XOR operation. The ciphertext c is then sent together with $E(even)$, and E is an encryption cipher. This scheme cuts the encryption cost by one half since the XOR operation is much simpler than the standard encryption cipher while achieving high security. An even faster algorithm called the permutation encryption was proposed by Chu *et al.* [22]. It treats a bit stream in the unit of bytes and performs a byte permutation according to a key. Since the permutation operation is much simpler than cryptographic operations, it has a fast speed. It is however a fixed byte-level permutation, which is shown to be vulnerable to the known-plaintext attack in [23].

The schemes presented above either require large computational complexity to achieve high security or provide poor protection to attacks at a lower computational cost. The design of an efficient yet secure bitstream-oriented encryption scheme is a challenging problem.

3.8.2 Statistical Properties of Compressed Bit Streams

Let us examine the generation process of a compressed multimedia bit stream so as to understand its structure better. Take video compression system as an example. The flow of most well known video compression algorithms can be stated as follows. The sequence of input

video frames first goes through several processing steps, including motion estimation, discrete Cosine transform (DCT) and quantization. The results of these processing steps, such as quantized DCT coefficients and motion vectors, are often called quantities or symbols. To further reduce the data size, these symbols are further compressed by a lossless entropy coder to produce a consecutive binary string consisting of 0's and 1's, which is called the bit stream.

Generally speaking, there is little statistical correlation between symbols. For example, the DCT operation is used to remove the spatial correlation while motion estimation/prediction is used to remove the temporal correlation. The remaining redundancy in symbols can be further removed by the entropy coder. That is, advanced entropy coding techniques such as adaptive Huffman coding and context-based adaptive arithmetic coding can squeeze out all remaining redundancy to make the output bit stream behave almost like a statistically random 0-1 binary sequence.

To conclude, the output bit stream from a high performance source coder can be viewed statistically random with its compressed domain symbols nearly correlation-free. This property has been confirmed by previous study on the statistical behavior of MPEG bit streams in [11]. Thus, the RPB scheme can be used to encrypt video bit stream. In next subsection, we conduct several statistical tests to evaluate the randomness of a compressed video bit stream.

3.8.3 Randomness Measurement of Compressed Video Bit Stream

The test video clip used in our experiment is the "Foreman" test sequence, that is publicly available from the H.264 standard working group. The first 10 frames are compressed to a bit stream of 23039 bytes. A complete counting of the first 20,000 bits of the bit stream gives the

statistics in Table 2.1, where the amount of 20000 bits is chosen for the purpose of the FIPS 140-1 test. We use n_0 and n_1 to denote the numbers of 0's and 1's, and n_{00}, n_{01}, n_{10} and n_{11} to denote the number of two consecutive bits 00, 01, 10 and 11, respectively. Furthermore, B_i represents the number of blocks (which are consecutive runs of 1's) of length i and G_i the number of gaps (which are consecutive runs of 0's) of length i. n_i denotes the number of occurrences of the i^{th} type of sequence of length $m, 1 \leq i \leq 2^m$.

n_0	n_1	n_{00}	n_{01}	n_{10}	n_{11}
10152	9848	5403	4749	4749	5099

Table 3.3: Counts of 1-bit and 2-bit subsequence.

We see from these data that the bit stream file possesses some characteristics of a random sequence. For example, n_0 roughly equals n_1, n_{00}, n_{01}, n_{10} and n_{11} are in a close range. To obtain a more accurate evaluation of the statistical randomness, we subject these 20000 bits to the FIPS 140-1 statistical test, which is specifically designed to measure the randomness level of a binary sequence. Due to space limitation, we only outline experimental results and briefly explain their meanings. For more theoretical treatment regarding the FIPS 140-1 statistical test, please refer to [13] at page 175-185.

1. *monobit test*

   The test is passed if the number of 1's satisfies $9654 < n_1 < 10346$. It is clear from Table 3.3 that the test bit stream passed the monobit test.

2. *poker test*

   The poker test determines whether sequences of length m each appear approximately the same number of times. The statistics used is

$$X = \frac{2^m}{k}\left(\sum_{i=1}^{2^m} n_i^2\right) - k, \quad k = \lfloor n/m \rfloor,$$

which should approximately follow the χ^2 distribution with $2^m - 1$ degrees of random. For the FIPS 140-1 test, $m = 4$, $k = 20000/4 = 5000$ is used to compute the value of X. The poker test is passed if $1.03 < X < 57.4$.

4-bit sequence	0000	0001	0010	0011	0100	0101	0110	0111
# of occurrence	364	343	323	292	309	282	322	283
	1000	1001	1010	1011	1100	1101	1110	1111
	322	292	273	298	374	250	304	369

Table 3.4: Poker test in the FIPS 140-1 statistical test.

The test bit stream barely fails the poker test as $X = 61.6$ is computed from n_i's in Table 3.4. One main reason is that byte value 0x00 and 0xFF are used in the compressed bit stream as a delimiter, leading to a highly biased number of occurrence. Please note that we have n_1 (number of "0000") equal to 364 and n_{16} (number of "1111") equal to 369, which deviate far from the expected number $k/m = 5000/16 = 312.5$.

3. *runs test*

   This test is passed if the 12 counts B_i and G_i, $1 \leq i \leq 6$, are each with in the interval specified in the following table. For purpose of this test, runs of length greater than 6 are counted as length 6. For this test bit stream, only 2 of the 12 counts are out of range. They are $G_1 = 2189 < 2267$ and $G_5 = 224 > 223$.

4. *long run test*

   This test is passed if there are no runs of length 34 or more. The longest run in the 20000 bits is G_{23} (a run of 23 0's) so the test bit stream passes the long run test.

Given the fact that these tests are specifically designed to check the quality of cryptographic pseudo-random bit generators, we may say that the compressed video bit stream obtained by the cutting-edge video compression system such as H.264 is *nearly* random. Thus, the use of the RPB encryption scheme for a coded multimedia bit stream can be justified.

Length of run i	Required interval	B_i	G_i
1	$2267 \sim 2733$	2324	2189
2	$1079 \sim 1421$	1210	1204
3	$502 \sim 748$	562	622
4	$223 \sim 402$	309	337
5	$90 \sim 223$	167	224
6	$90 \sim 223$	176	173

Table 3.5: Runs test in the FIPS 140-1 statistical test.

3.8.4 Entropy Measurement of Bit Streams A and C

As discussed in Sec. 3.6, the RPB encryption module does not alter the entropy of input stream A. In this section, we measure the entropies of the coded video bit stream A and the encrypted bit stream C. First, counts of subsequences in A and C are listed the following tables $3.6 \sim 3.8$.

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

Table 3.6: Counts of 1-bit and 2-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 3.7: Counts of 3-bit subsequences in A and C.

The entropies of order up to 4 are calculated and compared in Table 3.9. Note that the entropy measure is normalized by dividing the i-th 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 solid proof that A is a nearly random sequence.

2. For our test bit stream, 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, number of

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

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

Table 3.9: The entropy values of A and C with the order equal to 1, 2, 3 and 4.

3-bit subsequence "000" in A is 2843 while in C it is 2588, much 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, *i.e.,* it contains a highly biased number of byte value 0x00 and 0xFF as mentioned in the FIPS 140-1 poker test.

3. Thus for input with certain regular structures, RPB encryption in fact increases its randomness, as demonstrated by this particular test bit stream. This corroborates our claim that the RPB encryption in general does not weaken the randomness of its output.

3.9 Conclusion

A new encryption scheme designed for random plaintext data called the RPB scheme, was presented in this work. The input data are partitioned into random-sized blocks and a random rotation is performed in each block to achieve encryption in the RPB cipher. Security

study was conducted to show that both the ciphertext-only attack and the known/chosen-plaintext attack demand a computation complexity that grows exponentially with the size of the plaintext/ciphertext. Being efficient and highly secure, the proposed RPB scheme is suitable to secure data which are nearly random. One good application area is multimedia data encryption where a large amount of compressed audio/video data needs to be encrypted and decrypted in real time.

Chapter 4

Distortion-Aware Multimedia Authentication using Distance-Preserving Hash Function

4.1 Introduction

The cryptographic hash function provides a frequently used mechanism to protect the integrity of transmitted data. It aims at protecting the bit-wise integrity and, to be considered authentic, the received message must be exactly the same as the original one. For a good hash function, even a single bit modification to the input message leads to a much different hash output so as to be captured by the end user. However, in the context of multimedia data files, content integrity is usually more important than bit-wise integrity. They could be viewed as authentic if the content meaning is well preserved yet the bit stream is slightly modified. Many existing multimedia authentication schemes involve the use of a feature vector to represent the content meaning of a multimedia file and protect the integrity of the feature vector with either a hash function or a digital signature scheme. However, the problem of bit-wise sensitivity still remains. That is, if the hash of the received feature vector does not match with the received hash in the bit-wise sense, the only thing we can tell is that there is

some distortion occurring in the received data. However, we are not able to tell how close of the original and the distorted data. This is certainly not a desirable situation.

In this chapter, we address this issue by introducing a new type of hash function called the *Distance Preserving* (DP) hash function. Such a function has one additional characteristic. That is, by performing a certain operation on two hash values, one can obtain a distance measure between their corresponding inputs. Even when the two hash values do not match, we are able to get some additional information to evaluate the severity of distortion. Furthermore, the DP hash function is designed in such a way that the distance information is only accessible to the authorized party under the control of a secret key. The hidden distance information will not be leaked to anyone without knowledge of the correct key.

The rest of this chapter is organized as follows. The hash function and its use in data authentication are introduced in Sec. 4.2. Sec. 4.3 discusses the hard decision problem associated with regular hash function and proposes the idea of soft authentication. The concept of the DP hash function is developed in Sec. 4.4. In Sec. 4.5, the construction of a DP function based on a regular hash function is discussed. Sec. 4.6 investigates properties of the DP hash function. The security strength of the DP hash function is analyzed in Sec. 4.7. A generalized framework for soft multimedia authentication using the DP hash function is presented in Sec. 4.8. In Sec. 4.9, experiments are conducted to illustrate the use of the DP hash function in image authentication. Finally, concluding remarks are given in Sec. 4.10.

4.2 Background: Data Authentication using Hash Functions

The formal definition of a one-way hash function was first given in the seminal paper of Diffie and Hellman [24]. It is a fundamental concept used in nearly every aspect of modern cryptography: message authentication, public key cryptography, digital signature, etc. Furthermore,

the one-way function is an underlying cornerstone of cryptography, since it was closely related to many other cryptographic primitives. For instance, the existence of one-way functions is shown to be equivalent to the existence of secure digital signature schemes [25] and secure pseudo-random generators [26]. Although the existence of a one-way function has not been strictly established in theory, it is widely believed that such functions do exist. In fact, the practical implementation of one-way functions, *i.e.*, cryptographic hash functions, already exists for a long while. Generally speaking, the design of hash functions has been an active research topic in theory [27], [28], and they are widely used in practical security applications.

A hash function $h(\cdot)$ maps a binary string x of an arbitrary finite length to an output $h(x)$ of fixed length, say n bits. The output $h(x)$ is usually called a *hash-value* and the input x is referred to as the *preimage* of the hash-value $h(x)$. Let $\{0,1\}^n$ denote the space of all n-bit binary numbers (or binary strings). A cryptographic one-way hash function $h : \{0,1\}^\infty \longmapsto \{0,1\}^n$ can be formally defined as a mapping with the following properties [13].

1. *preimage resistance (one-way)*

   Given any hash-value y, it is computationally infeasible to find any input which hashes to that value, i.e., to find any preimage x such that $h(x) = y$.

2. *2nd preimage resistance*

   Given any input x, it is computationally infeasible to find any second input which hashes to the same output, i.e. to find another preimage $x' \neq x$ such that $h(x) = h(x')$.

3. *collision resistance*

   It is computationally infeasible to find any two distinct inputs $x' \neq x$ (a collision input pair) which hash to the same output, i.e., such that $h(x) = h(x')$.

Due to the above properties, a hash-value serves as a compact representative image (also called the digital fingerprint or the message digest) of the input message. Note that $h(\cdot)$ is by definition a many-to-1 mapping, which implies that *collisions* are unavoidable. In other words, there exist more than one input that hash to an identical output. However, the one-way property ensures that the original input message is difficult to obtain from the hash-value. Besides, the 2nd preimage resistance property makes it almost impossible to find any collision input that hashes to a predefined hash-value. Consequently, a hash-value can be viewed as if it were uniquely identifiable with its preimage. That is, if $h(x) = h(x')$, we will have $x = x'$ with overwhelming assurance and accept the received x' as authentic since the probability that x and x' happen to be a pair of collision input is negligibly small for a good cryptographic hash function $h(\cdot)$.

By appending a hash-value to transmitted data and comparing the hash-value of received data with the received hash, one can check the integrity of the received data. This is the classical approach used in data authentication.

4.3 The Hard Decision Problem, Information Loss and Soft Authentication

A fundamental question in data authentication based on the hash function is what happens if $h(x) \neq h(x')$? If this is the case, the only information a hash function can provide is that the received data x' must have been modified during transmission. However, since one cannot invert $h(x)$ to get the preimage x, there is no way to acquire any further information about the error by simply observing the hash-value $h(x)$ and $h(x')$. We can neither tell whether it is a single-bit or multiple-bits error nor are we able to determine the exact location of the

error. The entire x' is declared invalid and discarded regardless of what the actual error is. Thus, the use of a hash function gives rise to the "0-1" hard decision problem, where the received data are either totally accepted as authentic or totally discarded as invalid. There is no intermediate solution between these two extremes.

To better understand why hash functions lead to such extremes, we take a look at three processes: compression, encryption and hash. All of them can be considered as techniques to remove redundancy of any given input and produce a random output. In the previous chapter, we have already shown some interesting relationships between compression and encryption. Here, we consider the hash function, too, and make a more complete comparison among these three.

	Encryption	Compression	Hash function
Output size	same as input	fraction of input	fixed
Output randomness	highly random	nearly random	highly random
Length reduction	1:1	$10 \sim 40 : 1$	arbitrary:fixed
Redundancy removal	by random mapping	by length reduction	both of above
Reversibility	reversible	reversible	***irreversible***
Information loss	no	slight	***severe loss***

Table 4.1: Comparison of encryption, compression and hash.

We see from table 4.1 that hash functions do share similarities with encryption and compression in certain aspects. However, there are two fundamental characteristics that distinguish the hash function from the other two.

1. First, a hash function is *irreversible*, *i.e.*, given a hash-value, one cannot recompute the original input as done in encryption and compression. According to information theory, an information source can be represented at rate R only if R is not less than the entropy of the source. In compression, one attempts to make R as close to entropy as possible. However, a hash goes far beyond that extent because the output is *fixed-size* no matter what the actually input size is. As a result, the maximal information that a hash-value

is able to contain is less than the entropy of the preimage. Consider a 40 kilo-bytes file hashed by SHA-1 algorithm to a 160-bit hash-value. The length reduction 2048:1 is so severe that it is impossible to recover the original file.

2. The severe length reduction during a hashing process causes severe information loss, which is the reason of the "0-1" hard decision problem mentioned above. As a result, almost all information about the preimage is lost and a hash-value can only tell whether the received signal is modified or not but fails to provide any further information in case of errors. We emphasize that the information loss here is much severer than that in lossy compression. Take video coding as an example. Although lossy operations such as quantization lead to information loss, the visual quality degradation due to the loss is often small. Furthermore, this kind of information loss can be attenuated or even eliminated by proper settings, *i.e.*, using a smaller quantization step.

Such a hard decision is appropriate for certain applications such as authentication for highly-sensitive data. The received data is totally useless even if only 1 bit is modified, while the exact error is not important. This can be considered "hard" authentication: the meaning of authenticity is very strict at the bit-wise granularity. However, there are also applications where the authenticity of target data needs to be evaluated in a more flexible sense. If an error occurs, one wants to acquire more information regarding the exact error, rather than simply reject the whole data as unauthentic. Based on the error information, one can make a quantitative evaluation of the exact distortion and render an "authenticity degree", which is a value between 0 and 1 reflecting how similar the received data are to the original data. In contrast to hard authentication, this is a "soft" authentication: the meaning of authenticity is more flexible.

Multimedia data authentication is an application area where such a soft authentication scheme is desirable. When a multimedia file is transmitted, the file may be modified by legitimate operations such as transcoding and watermarking. The file will usually change a little bit after being processed by such operations. But it should be considered authentic as long as the multimedia content meaning is well preserved. Even if a malicious attack is applied to the file, a user still wants to observe the actual content quality and evaluate the severity of distortion. A key issue in designing such a soft authentication scheme is to have the authenticator yield additional indicator to reveal the authenticity degree. Clearly, regular hash functions fail to meet this requirement. New tools are needed to achieve this goal.

4.4 Distance-Preserving Hash Function

To solve the hard decision problem, it is desirable that outputs of hash function can carry additional information about the error between the two inputs. Since the error is generally the difference of two signals, it can be formulated by an appropriate distance measure in the input space. For instance, a 0-1 message string of length n can be treated as an element in the space composed of all n-bit unsigned binary integers. We can define several distance measures to model the error between two elements x and y in this space, such as the 1-norm $d(x,y) = |x - y|$ and the 2-norm $d(x,y) = |x^2 - y^2|^{1/2}$. More formally, a function $d(x,y)$ is called a distance, if it satisfies the following 3 conditions:

1. *Non-negativity*

   $\forall x$ and $\forall y$, $d(x,y) \geq 0$. $d(x,y) = 0$ if and only if $x = y$

2. *Transitivity*

   $\forall x$ and $\forall y$, $d(x,y) = d(y,x)$

3. *Triangle Inequality*

$\forall x$, $\forall y$ and $\forall z$, $d(x,y) + d(y,z) \geq d(x,z)$

We generalize the concept of the regular hash function and propose a new hash function that can embed in itself the distance between inputs. We call such a hash function the distance-preserving (DP) hash function, where the hash outputs preserve the distance between inputs, and denote it by $Hdist(\cdot)$. In order to reflect the distance between inputs, we would require the following condition hold:

$$\forall x \text{ and } \forall y, |Hdist(x) - Hdist(y)| = d(x,y), \text{ where } d(x,y) \text{ is a distance measure.} \quad (4.1)$$

However, a function satisfying Eq. (4.1) violates the one-way property of the hash function. To see why, let us consider a given hash-value z and let $h^{-1}(z)$ be the preimage of z. To compute $h^{-1}(z)$, one can choose a random element x in the input space, apply the hash and get $d(x, h^{-1}(z)) = |Hdist(x) - z|$. From x and $d(x, h^{-1}(z))$, the preimage $h^{-1}(z)$ can be computed. We would like to provide two more comments on Eq. (4.1). First, the arithmetic operation on the DP hash-value does not have to be a minus operation. There should be no restriction on this operation as long as it can extract the distance between inputs. Second, the error information is only intended for legitimate users. Any unauthorized third party is not allowed to obtain the distance information using a random input to Eq. (4.1) as shown in the above analysis.

To control access to the hidden distance information, our solution is to add a security key k as the second input to the DP hash function. It prohibits an unauthorized party from gaining the distance information. Only those who know the correct key can unlock the hash-value and obtain the distance information hidden therein. For anyone who does not know

the correct key, the DP hash function works virtually the same as a regular hash function. A formal definition of a DP hash function is given below.

Definition 7 *A distance-preserving* (DP) *function*

$$Hdist(x,k) : \mathbb{X} \times \mathbb{K} \longmapsto \mathbb{R}$$

is a functions satisfying the following distance-preserving property:

$$\begin{aligned} &\forall k_1 = k_2 \in \mathbb{K}, \text{ then } \forall x \text{ and } \forall y \in \mathbb{X}, f(\, Hdist(x,k_1), Hdist(y,k_2)\,) = d(x,y) \\ &\forall k_1 \neq k_2 \in \mathbb{K}, \text{ then } \forall x \text{ and } \forall y \in \mathbb{X}, \text{ there is no speicial relationship between} \\ &Hdist(x,k_1) \text{ and } Hdist(y,k_2) \end{aligned} \tag{4.2}$$

$Hdist(x,k)$ is further called a DP hash function if it is also a one-way hash function per input x and k. Notations are as follows:

$\mathbb{X}$ *: the space of input x*

$\mathbb{K}$ *: the space of input k*

$\mathbb{R}$ *: the space of DP function output*

$d(\cdot,\cdot)$ *: a distance measure in space $\mathbb{X}$*

$f(\cdot,\cdot)$ *: a binary arithmetic operation in space $\mathbb{X}$*

From the above definition, we see that key k does prevent the distance information being accessed by unauthorized outsiders. It is important that the content creator and verifier use a secret value only known to them as key k. In secure multimedia applications, the legitimate sender and receiver often share some secret information such as the session encryption key. This shared secret can be used as the key for the DP hash function.

4.5 Constructing Distance-Preserving Functions with Hash Functions

The proposed DP hash function is a general concept that attempts to capture the distance between inputs. The exact form of a DP hash function depends on what particular distance is used to measure the distortion. Our general idea to construct the DP hash functions is to use a random number to mask input x. For security protection, this random number is generated as $h(k)$, a one-way function output of key k. When two outputs using the same key are evaluated under the $f(\cdot,\cdot)$ operation, the same $h(k)$ cancel out and the difference between inputs is restored. If two different keys k_1 and k_2 are used, the two values $h(k_1)$ and $h(k_2)$ cannot be properly canceled so that the distance between inputs is still masked. In the following, we discuss several popular distances used in multimedia authentication and study their corresponding distance preserving (DP) functions. The difference between the DP function and the DP hash function will be explained in Sec. 4.6.

Example 1: the Hamming distance

The Hamming distance between two binary numbers x and y is defined as the number of bit positions where x and y differ. Let $\sum x$ denote the number of 1's of binary number x. The Hamming distance between x and y can be computed via

$$Hamming(x, y) = \sum(x \oplus y)$$

where $\oplus$ is the XOR operation. A DP function for the Hamming distance can be defined as

$$Hdist_{ham}(x, k) = x \oplus h(k) \tag{4.3}$$

Notice that XOR operation satisfies $x \oplus 0 = x$ and $x \oplus x = 0$ for any x. Thus, the XOR of two outputs using the same key k recovers the hamming distance between original inputs as

$$\begin{aligned}\sum(\, Hdist_{ham}(x,k) \oplus Hdist_{ham}(y,k)\,) &= \sum(\, x \oplus h(k) \oplus y \oplus h(k)\,) \\ &= \sum(\, x \oplus y \oplus h(k) \oplus h(k)\,) \\ &= \sum(\, x \oplus y \oplus 0\,) = \sum(\, x \oplus y\,) \\ &= Hamming(x,y)\end{aligned}$$

On the other hand, if two different keys k_1 and k_2 are used in the DP function of x and y, the XOR of two DP outputs is no longer equal to the hamming distance of the original inputs:

$$\begin{aligned}\sum(\, Hdist_{ham}(x,k_1) \oplus Hdist_{ham}(y,k_2)\,) &= \sum(\, x \oplus h(k_1) \oplus y \oplus h(k_2)\,) \\ &= \sum(\, x \oplus y \oplus h(k_1) \oplus h(k_2)\,) \\ &\neq Hamming(x,y)\end{aligned}$$

Example 2: the 2-norm distance

The 2-norm distance of x and y is defined as $2\text{-}norm(x,y) = |x^2 - y^2|^{1/2}$. It is also called the Euclidean distance. A DP function for the 2-norm distance can be written as

$$Hdist_{2-norm}(x,k) = x^2 + h(k) \tag{4.4}$$

It is clear that the difference between two DP outputs under the same key is equal to the 2-norm distance between inputs, *i.e.*,

$$\begin{aligned}|\, Hdist_{2-norm}(x,k) - Hdist_{2-norm}(y,k)\,| &= |\, x^2 + h(k) - (y^2 + h(k))\,| = |\, x^2 - y^2\,| \\ &= 2\text{-}norm^2(x,y)\end{aligned}$$

The distance between inputs x and y cannot be extracted if two different keys k_1 and k_2 are used in the DP function of x and y:

$$\begin{aligned} |\, Hdist_{2-norm}(x, k_1) - Hdist_{2-norm}(y, k_2) \,| &= |\, x^2 + h(k_1) - (y^2 + h(k_2)) \,| \\ &= |\, (x^2 - y^2) + (h(k_1) - h(k_2)) \,| \\ &\neq \textit{2-norm}^2(x, y) \end{aligned}$$

We have demonstrated in both cases that the two DP function outputs using the same key k preserves the distance between inputs, which can be recovered by a certain operations (*i.e.*, XOR for Hamming distance and subtraction for 2-norm distance). The use of two different keys k_1 and k_2 will lead to two random numbers $h(k_1) \neq h(k_2)$ in DP function outputs, thus effectively masking the distance information. For a good hash function $h(\cdot)$, the probability of $h(k_1) = h(k_2)$ (*i.e.*, k_1 and k_2 happen to a pair of collision inputs) is too small to be a real threat if the true key k_1 is not known to an attacker.

4.6 Properties of Distance-Preserving Hash Functions

When it comes to authentication, it is desirable that a DP function is also a cryptographic hash function. This is however not always true due to the inherent conflict between the hash and the distance. Two important questions arise here. First, for a given distance measure in the input space, it remains a question whether a DP hash function exists or not. Second, if a DP hash function does exist, is there a general construction method applicable to any distance measure? In this section, we attempt to provide some answers to these two problems.

For a DP function $Hdist(x, k)$, we denote by $\mathbb{X}$ the space of input x and $\mathbb{K}$ the space of input k. Some properties regarding $Hdist(x, k)$ are given in the following lemma.

Lemma 7 *Let*

$$Hdist(x,k) : \mathbb{X} \times \mathbb{K} \longmapsto \mathbb{R}$$

be a DP function. Then, we have

1. *The cardinality of domain* $\mathbb{X}$ *is not larger than that of range* $\mathbb{R}$, i.e.,

$$|\mathbb{X}| \leq |\mathbb{R}| \tag{4.5}$$

2. *Collision under the same key* k *is impossible,* i.e.,

$$\forall k \in \mathbb{K},\ \textit{there is no}\ x_1 \neq x_2 \in \mathbb{X},\ \textit{such that}\ Hdist(x_1,k) = Hdist(x_2,k) \tag{4.6}$$

Proof : For any given $k \in \mathbb{K}$, the domain of function $Hdist(\cdot,k)$ contains $|\mathbb{X}|$ elements while the range contains $|\mathbb{R}|$ elements. Assume to the contrary that $|\mathbb{X}| > |\mathbb{R}|$. Then, by the pigeon hole principle, there exist at least two elements in domain $\mathbb{X}$ being mapped to the same output in range $\mathbb{R}$. That is,

$$\exists\, x_1 \neq x_2 \in \mathbb{X},\ Hdist(x_1,k) = Hdist(x_2,k) = r \in \mathbb{R} \tag{4.7}$$

Since $Hdist(\cdot,\cdot)$ is a DP function, we have from Eq. (4.2) that

$$f(\, r,r\,) = f(\, Hdist(x_1,k), Hdist(x_2,k)\,) = d(x_1,x_2) \tag{4.8}$$

On the other hand, substituting $Hdist(x_1,k) = Hdist(x_2,k)$ into (4.8) gives

$$f(\, r,r\,) = f(\, Hdist(x_1,k), Hdist(x_1,k)\,) = d(x_1,x_1) = 0 \tag{4.9}$$

Finally, equating (4.8) and (4.9), we get

$$d(x_1, x_2) = 0 \text{ for } x_1 \neq x_2 \in \mathbb{X}$$

which contradicts the fact that $d(\cdot, \cdot)$ is a distance measure in input space $\mathbb{X}$. Hence, the assumption $|\mathbb{X}| > |\mathbb{R}|$ and derived Eq. (4.7) must be false. This completes the proof. ■

The two features in Lemma 7 reveal internal mechanisms regarding the distance-preserving property of a DP function. Because distance is a structured measure, a mapping that preserves distance must have the range not smaller than its domain. Furthermore distance is a unique-valued measure, thus two distinct inputs cannot be mapped to the same output. In contrast to regular hash functions, the following two salient features of a DP function are clearly observed.

1. It does not provide compression.

   The regular hash function compresses an input of an arbitrary finite length to an output of fixed length, say, n bits. In other words, $|\mathbb{X}| \gg |\mathbb{R}| = 2^n$. However, for a DP hash function, Eq. (4.5) tells us that $|\mathbb{X}| \leq |\mathbb{R}|$. Thus, the output of a DP hash function is at least as long as the input. It should be noted that the compression of an input is a desired feature in authentication.

2. There is no collision under the same key.

   For a regular hash function, a direct consequence of compressing a long input sequence to a short output sequence is the existence of collision inputs, *i.e.*, different inputs may hash to the same output. However, for a DP hash function, collision under the same key is not possible as proven by Eq. (4.6). However, this does not exclude the possibility of collision inputs under different keys.

4.7 Security Strength of DP Hash Function

Another important question regarding the DP function is whether it can be a hash function having the one-way property. Note that Lemma 7 is a necessary but not a sufficient condition for a DP hash function. A DP function satisfying Lemma 7 is not necessarily a one-way hash function. Our study suggests that this answer depends on the distance measure used and the particular format of the DP function.

For instance, the DP function for the Hamming distance $x \oplus h(k)$ as given in Eq. (4.3) is not a one-way hash function since finding a preimage is easy. For any given z, simply choose any $k \in \mathbb{K}$ and compute $x = z \oplus h(k)$. Then, (x, k) is a preimage of z. On the other hand, the DP function with the 2-norm distance

$$Hdist_{2-norm}(x, k) = x^2 + h(k) \tag{4.10}$$

is also a one-way hash function. In this section, we focus on this function and study its one-way property and security strength.

4.7.1 Preimage Attack Analysis

Lemma 8 *Consider the DP function*

$$g(x, k) = x^2 + h(k)$$

where $x \in \mathbb{X} = \{0,1\}^m$ *and* $h(k) \in \{0,1\}^n$. *The computational complexity of finding a preimage of* $g(x, k)$ *is*

$$\min(\ 2^n, \frac{2^{2m} - 2^{m+1} + 2^n}{2^m}\ )$$

Proof: Let $\mathbb{G}$ denote the range of $g(\cdot,\cdot)$. For $z \in \mathbb{G}$, a preimage attack aims at finding a pair of input x and k such that $g(x,k) = z$.

First, let us study the relationship between the lengths of x and k in $g(x,k)$, which are of m and n bits, respectively. Generally, we have $m < n$. Otherwise, if $m \geq n$, for all $x > 2^{n-1}$, $(x+1)^2 - x^2 = 2x + 1 > 2^n$. That is, the gap between successive x^2 is larger than the maximal value of $h(k)$, which enables one to determine the value of x for a large value of z. Practically, it is not difficult to construct a hash function with a longer output length from a basic hash function $h(\cdot)$ with a shorter output length, such as concatenating $h(k)$, $h(k+1)$, $h(k+2)$, $\cdots$. Thus, when $m \geq n$, we just extend the output length of $h(\cdot)$ until $m < n$ is satisfied.

The maximal element of space $\{0,1\}^m$ is $2^m - 1$. Thus, the maximal value of $z = x^2 + h(k)$ is $(2^m - 1)^2 + 2^n - 1 = 2^{2m} - 2^{m+1} + 2^n$ and any value of z less than this bound can be obtained for the reasons discussed above. Hence, $|\mathbb{G}| = 2^{2m} - 2^{m+1} + 2^n$. For a given $z \in \mathbb{G}$, we may consider the following two attack strategies.

1. Attack on $h(k)$

   A random m-bit number x is first chosen. The cryptanalyst then attempts to find k such that $h(k) = z - x^2$. Given the randomness of z and x, this is equivalent of finding a preimage of $h(\cdot)$ for a random value. The computational complexity is known to be 2^n for a good n-bit hash function.

2. Attack on x^2

   In this case, a random key $k \in \mathbb{K}$ is chosen and the cryptanalyst computes $z - h(k)$ and checks to see whether it is a square of some $x \in \{0,1\}^m$. The computational complexity is equal to the total number of trials before a hit is met.

Due to the randomness of z and $h(k)$, the value $z - h(k)$ can be regarded as a uniformly distributed random variable in the interval $[0,\ |\mathbb{G}|]$. Hence, the probability that $z - h(k)$ happens to be a square is

$$\text{Prob}\{\ z - h(k) = x^2\ :\ x \in \{0,1\}^m\} = \frac{|\{0,1\}^m|}{|\mathbb{G}|} = \frac{2^m}{2^{2m} - 2^{m+1} + 2^n} \tag{4.11}$$

Let p represent this probability, and t be the number of trials before the first success. Then, the above equation means

$$\text{Prob}\{t = 1\} = p$$

Since successive trials are independent (*i.e.*, the choice of k is random), we have

$$\text{Prob}\{t = k\} = (1-p)^k p$$

Thus, t follows the geometric distribution with probability p. It is a well known result that the expectation of t is

$$\text{E}(t) = 1/p = \frac{2^{2m} - 2^{m+1} + 2^n}{2^m}$$

Even though we do not exclude possibilities of other strategies or magic algorithms that might give a preimage of a given z, such a algorithm must be extremely hard to conceive of given the simplicity of $x^2 + h(k)$. Thus, the two strategies studied above seem to be the only effective algorithms. The computational complexity of a preimage attack is the minimum of two cases. This completes the proof. ■

Let us take a practical example to illustrate the difficulty of inverting the DP hash function (4.10). The hash function $h(\cdot)$ is the 160-bit SHA-1 algorithm. For several values of input length m, the computational complexity of finding a preimage is shown in Fig. 4.1. Note

that the y-axis is plotted in the logarithm to the base 2 scale. From this figure, we see that for $m < \frac{n}{2} - 4$, the computational complexity is equal to 2^{n-m}. While for $m > \frac{n}{2} + 4$ it becomes 2^m. The worst-case computational complexity is $2^{n/2+1}$, which occurs when $m = \frac{n}{2}$.

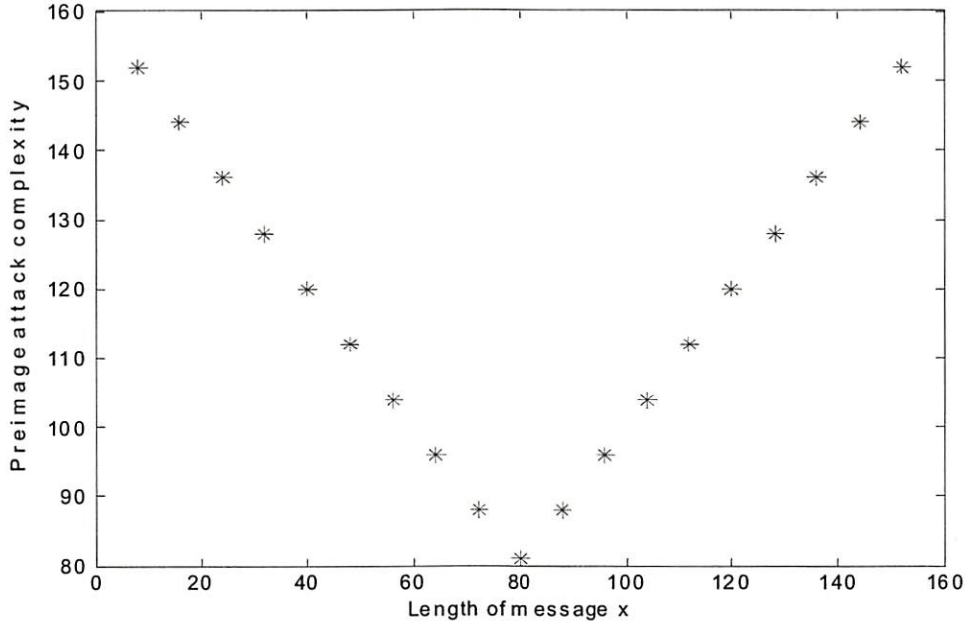


Figure 4.1: Preimage attack complexity of $g(x,k) = x^2 + h(k)$, $h(\cdot)$ is the 160-bit SHA-1 hash

It is well known that, for a good n-bit cryptographic hash function, the strength against the preimage attack is 2^n while the strength against the collision attack (*i.e.* finding a pair of collision inputs) is $2^{n/2}$. Thus, the DP hash function $x^2+h(k)$ in (4.10) yields a weaker security strength than the regular hash function $h(\cdot)$. Its worst-case security against the preimage attack roughly equals to the strength of $h(\cdot)$ against the collision attack. This reduced security strength can be viewed as the price to achieve the distance-preserving property.

4.7.2 Collision Attack Analysis

Although Lemma 7 establishes that there are no collision inputs under the same key for the DP function, it is possible that collision inputs under *different* keys exist, *i.e.*, $Hdist(x_1, k_1) =$

$Hdist(x_2, k_2)$ with $k_1 \neq k_2$. Our study on the DP hash function (4.10) has yielded the following results.

Lemma 9 *Consider the DP function*

$$g(x, k) = x^2 + h(k)$$

where $x \in \mathbb{X} = \{0,1\}^m$ *and* $h(k) \in \{0,1\}^n$. *The computational complexity of finding a pair of collision inputs* $(x_1, k_1), (x_2, k_2)$ *such that* $x_1^2 + h(k_1) = x_2^2 + h(k_2)$ *is*

$$\min(\sqrt{2^{2m} - 2^{m+1} + 2^n}, \frac{2^n - 1}{d})$$

where d *is the number of distinct values of* $x_1^2 - x_2^2$ *where* $x_1, x_2 \in \{0,1\}^m$ *and* $x_1^2 - x_2^2 \leq 2^n - 1$.

Proof: Along the same line of reasoning in the proof of Lemma 8, we claim that the size of $g(x, k)$'s range is $|\mathbb{G}| = 2^{2m} - 2^{m+1} + 2^n$. We consider the following two attack strategies.

1. Classical birthday attack

   The classical birthday attack on regular hash functions can be applied to the current case as well. The birthday attack comes from the occupancy distribution in statistics. Suppose that elements are randomly drawn from a set of N elements one at a time, with replacement, the expected number of draws that a repeated element will be encountered is $O(\sqrt{N})$. Here, we have $N = 2^{2m} - 2^{m+1} + 2^n$ elements. Hence the attack complexity to find collision inputs is $\sqrt{2^{2m} - 2^{m+1} + 2^n}$.

2. Attack on difference of x^2

   Note that if $x_1^2 + h(k_1) = x_2^2 + h(k_2)$, $h(k_2) - h(k_1) = x_1^2 - x_2^2$ is the difference of two

squared numbers. A wiser attack exploiting this fact goes as follows. We can choose two random keys $k_1 \neq k_2$, compute $h(k_2) - h(k_1)$ (or $h(k_1) - h(k_2)$ if $h(k_1) > h(k_2)$) and check whether it happens to be the difference between the two squared numbers. The attack complexity is equal to the total number of trials before a hit is met.

Since $h(\cdot)$ is an n-bit hash function, the maximal value of $h(k_2) - h(k_1)$ is $2^n - 1$. Due to the randomness of k_1 and k_2, the value $h(k_2) - h(k_1)$ can be regarded as a uniformly distributed random variable in interval $[0,\ 2^n - 1]$. Let d be the number of distinct values of $x_1^2 - x_2^2$, where $x_1, x_2 \in \{0,1\}^m$ and $x_1^2 - x_2^2 \leq 2^n - 1$. The probability that $h(k_2) - h(k_1)$ happens to be the difference between two squared numbers is

$$\text{Prob}\{\ h(k_2) - h(k_1) = x_1^2 - x_2^2\ :\ x_1, x_2 \in \{0,1\}^m\} = \frac{d}{2^n - 1}. \tag{4.12}$$

Let p represent this probability, and t be the number of trials before the first success. Again, along the same line of reasoning in the proof of Lemma 8, t follows the geometric distribution with probability p and the expectation of t is

$$\text{E}(t) = 1/p = \frac{2^n - 1}{d}.$$

The two strategies studied above seem to be the only effective algorithms given the simplicity of the construction $x^2 + h(k)$. The computational complexity of a collision attack is the minimum of two cases. This completes the proof. ■

Although we are not able to compute the accurate value of d at this time, we have the following rough estimate. Consider the m-bit number x with $m < n/2$. The maximal value of x^2 is $(2^m - 1)^2 < 2^n$. Hence, any x_1, x_2 will satisfy $x_1^2 - x_2^2 \leq 2^n - 1$. The number of possible values of $x_1^2 - x_2^2$ is approximately equal to $d = \binom{2^m}{2} \approx 2^{2m-1}$. Note that there are

repetitions of these values, for instance $7^2 - 5^2 = 5^2 - 1^2$ so that d will be smaller than that. For $m \approx n/4$, we have $d \approx 2^{n/2-1}$ and $\frac{2^n-1}{d} \approx 2^{n/2}$. As m grows, d will become larger. We conjecture that the collision attack complexity $\frac{2^n-1}{d}$ will drop below $2^{n/2}$.

It is well known that the collision attack complexity of a good n-bit cryptographic hash function is $O(2^{n/2})$. Thus, the DP hash function $x^2 + h(k)$ is more susceptible to the collision attack than a regular hash function $h(\cdot)$. This reduced security is attributed to the structured term x^2 and can be viewed as a tradeoff in order to attain the distance-preserving property.

4.8 Soft Multimedia Data Authentication Scheme

Based on the DP hash function, a generalized soft authentication framework is developed for soft multimedia authentication using feature extraction. The diagram of the proposed system is shown in Fig. 4.2. We describe system components and their corresponding roles in the soft authentication scheme following the message flow from the sender to the receiver.

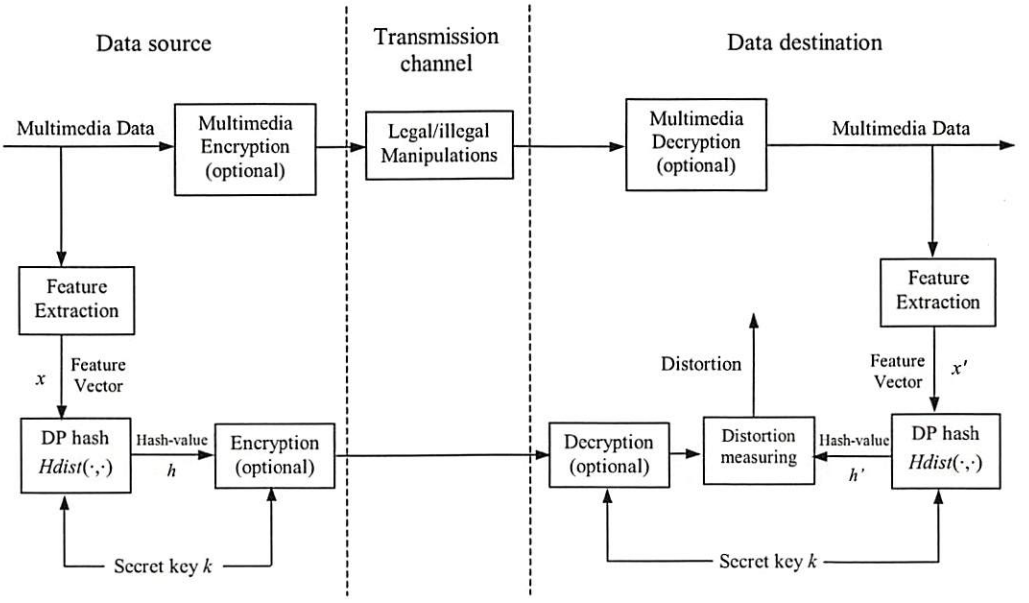


Figure 4.2: Soft multimedia authentication using the DP hash function.

A. The Sender (including the content creator)

A feature extraction algorithm is performed to produce a content feature vector $x = (x_1, x_2, \ldots, x_n)$ for the multimedia data, where each x_i is a component describing a different feature. Our scheme does not demand a particular feature extraction algorithm as long as one requirement is met. That is, the derived feature vector should reflect the meaning of multimedia content. Due to distortions in the transmission channel, the feature vector x' extracted at the receiving side is often different from the original x. Although the exact error between x and x' varies with different feature extraction algorithms, such an error reflects quality degradation of the received multimedia content. Depending on multimedia types and application requirements, an appropriate distance measure is used to model the distortion in the multimedia content for each feature of interest. The corresponding DP hash function on the derived feature vector x is computed. The second input k is a shared secret established between the sender and the receiver, *e.g.,* the session encryption key. The result is a group of DP hash values $Hdist(x_i, k), i = 1, 2, \ldots, n$ with a distance information embedded.

B. The transmission channel

The content creator sends the multimedia data and the DP hash-value $Hdist(x_i, k)$ through the channel. They can either be encrypted or of the plaintext format during transmission. The multimedia content is subject to manipulations, either legitimate processing by intermediate sites along the path or malicious tampering attack, before it reaches the intended recipient.

C. The receiver (including the content verifier)

The recipient receives the multimedia content as well as the DP hash-value $Hdist(x_i, k)$ from the transmission channel. The same feature extraction algorithm used in the sender is

adopted to generate a content feature x' of the received multimedia data. In most cases, x' will differ from the original x, and the error between them reflects the distortion occurred to the received multimedia data. With the shared key k, the DP hash-value $Hdist(x'_i, k)$ of content feature x' is calculated. By performing a proper operation on the received DP hash values and $Hdist(x'_i, k)$, we can obtain the distortion of the received multimedia data as the distance between x and x'

$$d(x_i, x'_i) = f(\, Hdist(x'_i, k), Hdist(x_i, k)\,),\ i = 1, 2, \ldots, n \tag{4.13}$$

The final step of the authentication scheme is to derive a degree of authenticity using the above distortion. For each component feature x_i, a authenticity degree $auth_i$ is computed using various methods depending on the particular distance measure.

For instance, if the Hamming distance is used, users might be interested in the percentage of the modified bits with respect to the total length. Then, the authenticity degree for component feature x_i can be defined as

$$auth_i = 1 - \frac{d(x_i, x'_i)}{|x|} \tag{4.14}$$

If the 2-norm distance is used for component feature x_j, then the corresponding authenticity degree is related to the error energy:

$$auth_j = 1 - \frac{d(x_j, x'_j)}{x_j^2} \tag{4.15}$$

where $auth_i$ is a value between 0 and 1 reflecting the authenticity degree of the multimedia data with respect to each component feature.

If there is no or small distortion so that the content feature is not affected, we have $x_i = x_i', d(x_i, x_i') = 0$ and $auth_i = 100\%$, meaning that the received content is authentic with respect to feature x_i. As distortion $d(x_i, x_i')$ grows larger and larger, $auth_i$ becomes smaller and smaller, indicating a decrease of content authenticity. Finally, by combining all $auth_i$'s, a user can render a soft authentication decision.

Several features of the proposed soft multimedia authentication system are summarized as follows.

1. *Applicability*

   The proposed system provides a general framework for multimedia data authentication based on the feature vector. It can accommodate various feature vectors based on the premise that the feature vector error can reflect the distortion of the received multimedia data.

2. *Flexibility*

   Our framework allows great freedom in implementing the particular authentication scheme. Depending on different authentication requirements and meaning of content authenticity, a designer can choose different distance measures to model the distortion of the component feature of interest. In addition, the definition of authenticity degree is also flexible. Different formula can be used to compute the authenticity degree according to application needs.

3. *Security*

   The DP hash function ensures that the error between inputs are only accessible to legitimate users knowing the secret key. For any unauthorized party, the DP hash-value appears just like a random number and does not disclose any useful information.

4.9 Simulation Results

In this section, experiments are conducted to demonstrate the use of the DP hash function in image authentication. We use the feature extraction scheme described in [29], where quantized DCT coefficients of two 8x8 blocks are compared to obtain bit "1", if the result is larger, and bit "0", otherwise. The location of blocks to be compared is determined by a random sequence generated from a secret key. It was claimed in [29] that this relationship is only changed by malicious manipulations but not by common image processing operations.

The image used in our experiment is the classical "Lena" image of size 512x512. The size of the obtained feature vector is 4096 bits. For simplicity, we use the DP function with a Hamming distance $Hdist_{ham}(x, k) = x \oplus h(k)$ as in (4.3) to encode feature vector x, where k is the secret key producing the location selection sequence and set to the string "Lena-experiment" in our simulation. $h(\cdot)$ is the SHA-1 160-bit hash function, repeatedly applied to k until we get 4096 bits. The computed DP hash output of the original Lena image is shown in Fig. 4.3.

Fig. 4.5(a) shows a deliberately modified Lena image where an additional decorative flower is added on the hat and one birthmark painted above the left eye. The DP hash output of this modified image is given in Fig. 4.4. We observe a 20-bit difference from the original hash, which are pointed by small vertical arrows. Due to space limitation, only segments of DP hash values where changes occur are shown, rather than the entire 4096 bits. The omitted parts of two DP hash values as represented by ... are identical.

A legitimate user knowing secret key k is able to compute the DP hash of the modified image and, in turn, detect the location of distortion. However, an unauthorized party cannot tell whether the image is authentic or not, even provided with the original DP hash in Fig. 4.3. He/she might accept the manipulated image, believing that Lena has a birthmark

```
b79ff8371f2b3cdca79f872df73dcdb2dd6f854275dc3bfb29bbe1ca61
...
ab2837ed625fe6f646be3d540e6bb723f0ac75c98330ace3f9f52fae97
...
c6fb6a57ad57a569c74d91d435d37a8d6968a013fd8709e167909c21fc
7b0bc95c0c6c591efda15ea2dcb2d37384aeac
```

Figure 4.3: The DP hash of the original Lena image.

```
b79ff8371f2b3cdca79f872df73dcdb2dd6f854275dc3bfb29bbe1ca61
...                                  ↓↓                 ↓
ab2837ed625fe6f646be3d540e6bb73bf0ac75c98330acebf9f52fae97
...                                                ↓        ↓
c6fb6a57ad57a569c74d91d435d37a8d6968a013fdc709e167908c21fc
7b0bc95c0c6c591efda15ea2dcb2d37384aeac
```

Figure 4.4: The DP hash of modified Lena image: difference marked by arrow.

above the left eye and two flowers on her hat. In Fig. 4.5(b), the manipulated region is correctly detected by the white mosaic tiles, exposing the fake flower and birthmark. Thus, this experiment demonstrates that the DP hash function is capable of identifying the exact distortion in a secure way only accessible by authorized users.

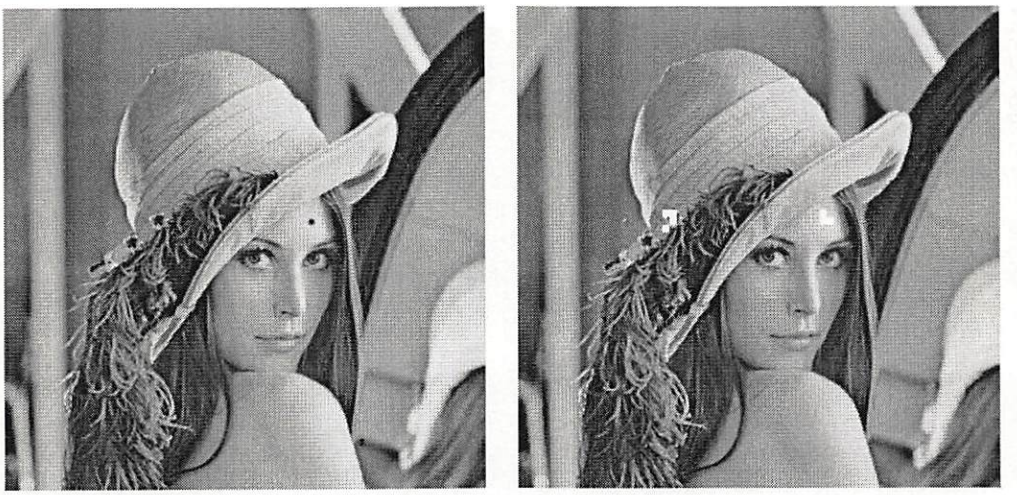


Figure 4.5: (a) The modified Lena image and (b) the location of detected distortion.

4.10 Conclusion

A new type of hash functions called the distance-preserving (DP) hash function is proposed and studied in this research. It has a special characteristic. That is, under the same security key k, a certain operation on outputs yields the distance between inputs. While under different k values, it appears indistinguishable from a conventional hash function. If the data have been tampered with, such a function can provide the distortion information between the received data and the original data in a secure way that is only accessible to the authorized party. We study the property of the DP hash function and give an example for its construction. Furthermore, a soft multimedia authentication scheme was developed where the DP hash function is used to encode the feature vector of the multimedia content. A legitimate user can obtain the feature vector distortion and compute the authenticity degree. Based on that information, the overall quality degradation can be evaluated.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

This research work has investigated the problem of light-weight multimedia encryption for resource-limited systems/devices and soft multimedia authentication. Several interesting findings haven been obtained and are summarized below.

A randomized entropy coding approach for multimedia data encryption was proposed in Chapter 2. The basic idea is to apply multiple entropy coding parameters (settings) dynamically according to a statistically random sequence to encode consecutive inputs. Based on the REC approach, three specific encryption schemes were developed using the Huffman, the arithmetic, and the LZ78 coders as the underlying entropy coders, respectively. The REC encryption approach only involves the selection of coding parameters in a random manner. Thus, it has a low computational cost and can be easily integrated into a multimedia compression system by either software or hardware. It does not impair the coding gain of the original data compression scheme. The REC method was proved to be secure under the ciphertext-only attack. Its resistance to known/chosen-plaintext attack is good due to the existence of multiple alias keys, which encrypt a given plaintext to the same ciphertext. With

these features, the REC method is suitable for multimedia encryption in systems or devices with limited resource/power.

The problem of efficiently and securely encrypting nearly random data was investigated in Chapter 3. This problem has both theoretical interest and practical significance due to wide availability of such data, *e.g.* compressed video and audio bit streams. A light-weight cipher called the RPB (Rotation in Partioned Blocks) scheme was proposed, which demands a low encryption cost and maintains high security strength. We also introduced a concept called alias keys that encipher a plaintext to an identical ciphertext by different keys, and study the existence and the number of alias keys. Security analysis showed that the proposed RPB cipher can withstand both ciphertext-only and known/chosen plaintext attacks since the computational complexity of these two attacks grows exponentially with the size of plaintext/ciphertext without the right key information. It was also demonstrated that entropies of the input and the output of the RPB cipher remain the same. Thus, the RPB cipher preserves the randomness of its input. Overall, the proposed RPB cipher can achieve the security strength of a block cipher at the encryption cost of a stream cipher. The high efficiency and security of the RPB cipher makes it an ideal choice in secure multimedia applications, where a large volume of compressed data has to be encrypted/decrypted in real time.

A new type of hash function $Hdist(x, k)$ called the distance-preserving (DP) hash function was proposed in Chapter 4. If the data are modified, such a function can provide the distortion information between the received data and the original data in a secure way that is only accessible to the authorized party. This is made possible by including key k as the input to the hash function. Under the same key k, the output of the DP hash function yields the distance between the original and the received inputs. Under a different key k', the DP hash

function is indistinguishable from the conventional hash function. Then, a soft multimedia authentication scheme was developed, where the DP hash function is used to encode the feature vector of the content. A legitimate user can obtain the feature vector distortion and compute the authenticity degree. Based on the information, the overall quality degradation can be assessed.

5.2 Future Work

Some future research topics will be conducted to complete the thesis study. They are described below.

5.2.1 Improvement to Current Results

There is room to further improve the current work to get deeper results in both theory and practice. Some specific problems to be studied include the following.

1. Alias key space search for the RPB scheme

   For the RPB scheme, we showed that the number of alias keys grows exponentially with the ciphertext size. However, we do not elaborate on the details about how to find multiple alias keys. For a given plaintext/ciphertext pair, an algorithm to find all its alias keys is an interesting one to explore. Both the existence and complexity analysis of such an algorithm are important to the security of RPB scheme under the known/chosen-plaintext attack.

2. Existence of the DP hash function

   We proposed the DP hash function to capture the distance of signals in the input space and established several properties of such a function such as its range cardinality and

collision under the same key. An important problem to be answered is that, for any distance in the input space, whether there exists a corresponding DP hash function.

3. General construction of DP hash function

   If the DP hash function exists for a particular distance measure, we would like to develop a general algorithm to construct the corresponding DP hash function. We have provided an example for the 2-norm distance in Chapter 4. However, a general construction method for any distance is still lacking at this time.

4. Collision attack complexity analysis of the 2-norm DP hash function

   We have shown that the complexity of the collision attack for the 2-norm DP hash function $x^2 + h(k)$ is inversely proportional to the parameter d, which is the number of distinct values of $x_1^2 - x_2^2 < 2^n - 1$. It was also conjectured that this complexity will be less than $2^{n/2}$, which is the complexity to find collisions for a regular hash function. A study to determine the size of d will give more conclusive results on this problem.

5. The large error differentiation problem of DP hash function

   The DP hash function provides some extra information about the error of transmitted data. If the error is relatively small, the corresponding distortion can be identified efficiently. However, if a large error is detected, it may be caused by a large distortion or due to a wrong key to extract the error. How to differentiate these two situations is an important problem.

5.2.2 Implementation on Embedded Systems and Performance Profiling

Our research aims at light-weight high security multimedia encryption schemes for resource-limited applications such as embedded systems and/or portable devices. We would want to

implement our proposed schemes in such an environment to evaluate its performance. This may include the following specific tasks.

1. Practical implementation on embedded systems

   We will choose an embedded system platform, *e.g.* the ARM-9 processor, to implement the proposed REC and RPB encryption schemes. The system performance such as the CPU and memory usage requirements will be evaluated.

2. Performance optimization and profiling

   We will attempt to optimize the implementation with respect to the encryption speed, the memory usage, the code size, etc. The result may lead to several customized variations with diverse performance profiling.

3. Performance comparison with cryptographic ciphers

   We will implement traditional cryptographic block and stream ciphers and compare their performance with the light-weight secure encryption schemes proposed in our work.

Bibliography

[1] H. Cheng and X. Li, "Partial encryption of compressed images and videos," *IEEE Trans. on Signal Processing*, vol. 48, pp. 2439–2451, Aug 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)*, Sep 9-11 2002.

[3] A. Pommer and A. Uhl, "Selective encryption of wavelet-packet encoded image data," *ACM Multimedia Systems (Special issue on Multimedia Security)*, vol. 9, pp. 279–287, 2003.

[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*, Oct 4-7 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 *4th ACM International Conference on Computer Communications and Networks*, Sep 20-23 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*, Feb 1996, pp. 137–144.

[7] L. Tang, "Methods for encryption and decrypting mpeg video data efficiently," in *4th ACM International Conference on Multimedia*, Nov 18-22 1996, pp. 219–230.

[8] C. Shi and B. Bhargava, "A fast mpeg video encryption algorithm," in *6th ACM International Conference on Multimedia*, Sep 1998.

[9] C. Shi, S.-Y. Wang, and B. Bhargava, "Mpeg video encryption in real-time using secret key cryptography," in *1999 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*, Jun 28 - Jul 1 1999.

[10] C. Wu and C.-C. J. Kuo, "Efficient multimedia encryption via entropy codec design," in *SPIE International Symposium on Electronic Imaging*, Jan 2001.

[11] L. Qiao and K. Nahrstedt, "A new algorithm for mpeg video encryption," in *Proc. of the 1st International Conference on Imaging Science, Systems, and Technology*, Jul 1997, pp. 21–29.

[12] C. Wu and C.-C. J. Kuo, "Fast encryption methods for audiovisual data confidentiality," in *SPIE Photonics East - Symposium on Voice, Video, and Data Communications*, Nov 2000.

[13] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptographyn.* CRC Press, 1996.

[14] I. H. Witten and J. G. Cleary, "On the privacy afforded by adaptive text compression," *Computers and Security*, vol. 7, pp. 397–408, 1988.

[15] H. A. Bergen and J. M. Hogan, "A chosen plaintext attack on an adaptive arithmetic coding compression algorithm," *Computers and Security*, vol. 12, pp. 157–167, 1993.

[16] S. A. I. J. G. Cleary and I. Rinsma-Melchert, "On the insecurity of arithmetic coding," *Computers and Security*, vol. 14, pp. 167–180, 1995.

[17] J. Lim, C. Boyd, and E. Dawson, "Cryptanalysis of adaptive arithmetic coding encryption scheme," in *2nd Australasian Conf. on Information Security and Privacy*, pp. 216–227.

[18] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. on Information Theory*, vol. 23, pp. 337–343, May 1977.

[19] ——, "Compression of individual sequences via variable-rate coding," *IEEE Trans. on Information Theory*, vol. 24, pp. 530–536, Sep 1978.

[20] A. Biryukov, "Block ciphers and stream ciphers: The state of the art," 2003.

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

[22] H.Chu, L. Qiao, and K. Nahrstedt, "A secure multicast protocol with copyright protection," in *Proc. of SPIE's Symposium on Electronic Imaging: Science and Technology*, 1999.

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

[24] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Trans. on Information Theory*, vol. 22, pp. 644–654, Nov 1976.

[25] J. Rompel, "One-way functions are necessary and sufficient for secure signatures," in *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, 1990, pp. 387–394.

[26] R. Impagliazzo, L. Levin, and M. Luby, "Pseudo-random generation from one-way functions," in *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, 1989, pp. 12–24.

[27] M. Naor and M. Yung, "Universal one-way hash functions and their cryptographic applications," in *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, 1989, pp. 33–43.

[28] M. Bellare and P. Rogaway, "Random oracles are practical: a paradigm for designing efficient protocols," in *Ist ACM Conference on Computer and Communications Security*, Nov 1993, pp. 62–73.

[29] C. Lin and S. Chang, "A robust image authentication method distinguishing jpeg compression from malicious manipulation," *IEEE Trans. on Circuits and Systems of Video Technology*, vol. 11, pp. 153–168, Feb 2001.

Appendix A

Proof of Theorems

A.1 Proof of Lemma 1

Proof : For any given bit stream C, we can parse C using the parsing attack algorithm described in Sec. 2.6.2. We view any $L+1$ consecutive integers $s, s+1, \ldots s+L$ as a window of length $L+1$ and examine the corresponding bit positions in this window as shown in Fig. A.1. Since L is the maximal code length of all Huffman tables used in RHT encryption, the parsing position must fall into this window at least once. Hence, there is a viable parsing for at least one length $s+\alpha$ between s and $s+L$. According to Def. 3 we have $V(s+\alpha) \geq 1$, $0 \leq \alpha \leq L$, which is the one given by (2.3).

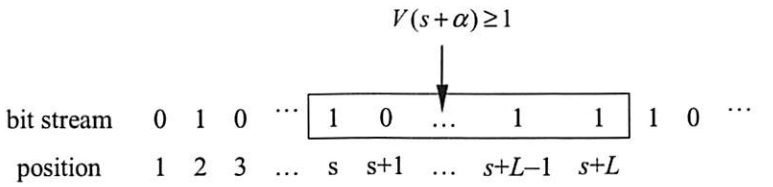


Figure A.1: Illustration of the viable parsing window.

Let $P(k)$ denote all viable k-parsing schemes. Suppose the last parsed code length is β_i. Then, it is clear that any parsing scheme in $P(k)$ comes from a viable $k-\beta_i$-parsing plus the last parsed code $c_{k-\beta_i+1}\, c_{k-\beta_i+2} \ldots c_k$, ending at k-th bit c_k. Thus, $P(k)$ can be classified according to the last code $c_{k-\beta_i+1}\, c_{k-\beta_i+2} \ldots c_k$ with length β_i. Note that for different β_i, the corresponding viable parsing schemes are distinct. For example, a viable $k-2$-parsing plus 2-bit code "11" is different from a viable $k-4$-parsing plus 4-bit code "1011", though both of them lead to a viable k-parsing ending at bit c_k. This is illustrated in Fig. A.2. Therefore, this categorization of $P(k)$ is all-inclusive and mutual-exclusive. The number of all viable k-parsings is the sum of all viable $k-\beta_i$-parsings. That is

$$V(k) = \sum_{\beta_i} V(k-\beta_i)$$

which is given by (2.4).

Note that Eq. (2.4) is a recursive relationship that can be applied to $V(k-\beta_i)$ as well. Starting at bit position k, $V(k)$ is represented as a sum of $V(k-\beta_i)$. By repeatedly applying Eq. 2.4 backward, $V(k)$ can be reduced to a weighted sum of more previous viable parsing $V(i)$'s with smaller length i. Due to the same reason, in the proof of Eq. (2.3), the length

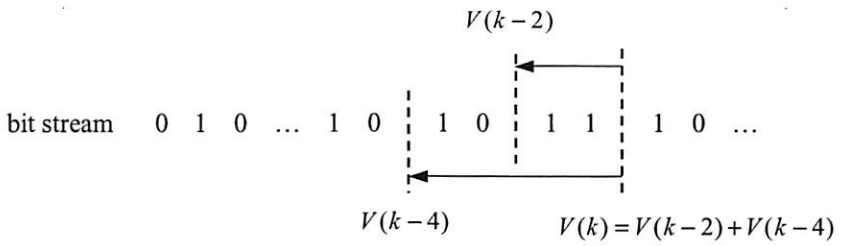


Figure A.2: The recursive relationship of $V(k)$.

i must fall into the interval $S = \{s, s+1, \ldots s+L\}$, $s+L \leq k$ containing $L+1$ integers at a certain point. We have $V(k) = \sum_i a_i V(i)$, $i \in S$ with appropriate coefficients a_i. This completes the proof. ■

A.2 Proof of Lemma 2

Let $A = (a_1 a_2 \ldots a_N)$ be a stream of N bits. Assume B is the maximum block size allowed in partitioning A. Notice that this restriction implies 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;

⋮

B) 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 bit stream $A' = RP(A, p, r)$ must belong to one and only one of the above categories. Thus, we have

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

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 $2 \leq k \leq B$ 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 $a_2 a_1$), 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 $a_2 a_3 a_1$) 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:

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

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

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

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

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

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

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

If we define the following recursive sequence

$$S(N) = \begin{cases} 2S(N-1) & N > 0 \\ 1 & N = 0 \end{cases}$$

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

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.3 Proof of Lemma 3

Let $C(N)$ to 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 A' such that at least $\lceil R(N)/C(N) \rceil$ RPB operations transform A to A'. That is, we have

$$Alias(A, A') \geq \lceil R(N)/C(N) \rceil$$

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 A' 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 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 A'. Thus, beginning from bit position $z_1 + B$ to the end, A' can contain at most $Z - 1$ 0's. Likewise, A' 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 2, we know that $R(N) > 2^N$. Hence, we conclude

$$Alias(A, A') \geq \lceil R(N)/C(N) \rceil > \left\lceil 2^N \Big/ \binom{N}{Z} \right\rceil$$

which is Eq. (3.3).

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

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

Finally, we end up with

$$Alias(A, A') > 2^N \Big/ \frac{2^N}{\sqrt{\pi N/2}} = \sqrt{\pi N/2}$$

This completes the proof. ■

A.4 Proof of Lemma 4

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

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

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

$$A(mN) \geq A^m(N), \quad m \text{ is an integer.} \tag{A.4}$$

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 3 that $A(N) > \sqrt{\pi N/2}$ for at least one ciphertext. Therefore, $c > 1$ when N is sufficiently large (otherwise, $A(N) \to 0$). This completes the proof. ■

A.5 Proof of Lemma 5

Let K_i denote the set of alias keys corresponding to each plaintext/ciphertext pair (A_i, C_i), $i = 1, 2, \ldots$. According to Lemma 4, each K_i contains about $A(N) \sim c^N$ keys out of all possible $R(N)$ keys. We use X_i to denote the set of possible keys. First, it is clear that $X_1 = K_1$. After each pair (A_i, C_i) is checked, the set of possible keys is the intersection of the current set and K_i; namely,

$$X_i = X_{i-1} \cap K_i$$

or, equivalently,

$$X_i = K_1 \cap K_2 \cap \cdots \cap K_i.$$

We would like to find out index i such that $|X_i| = 1$, *i.e.*, X_i contains only one element, which is the correct key. Instead of treating X_i directly, we consider its complementary set $\bar{X}_i = R - X_i$, where R denotes the set of all possible $R(N)$ keys as specified in Lemma 2. According to the set operation rule, we have

$$\bar{X}_i = \overline{K_1 \cap K_2 \cap \cdots \cap K_i} = \bar{K}_1 \cup \bar{K}_2 \cup \cdots \cup \bar{K}_i, \tag{A.5}$$

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

This problem turns out to be 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 result that the expected number of draws to collect all N coupons is $N \ln N$. The difference here is that the draw size is $R(N) - A(N)$ instead of 1. Due to the randomness of available plaintexts (note that a 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 an equal probability. Therefore, we can treat each 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 $R(N) - A(N)$, *i.e.*,

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

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 $\frac{R(N)}{R(N)-A(N)} \to 1$. Finally, taking 2^N as a rough estimate of $R(N)$, we obtain

$$P(N) \approx N \ln 2$$

which completes the proof. ■

A.6 Proof of Lemma 6

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

$$H^{(n)}(S) = \sum_{d} p(d) \frac{1}{\log p(d)}, \tag{A.6}$$

where the summation is over all possible n-bit subsequence d. For a sufficiently long sequence s, the probability of a particular subsequence d can be calculated as the number of occurrence of d divided by the length of s, *i.e.*,

$$p(d) = \frac{\#\text{ of occurrence of } d \text{ in } s}{\text{length of } s}. \tag{A.7}$$

For instance, the first- and the second-order entropies are computed by

$$H^{(1)}(S) = p(0) \log \frac{1}{p(0)} + p(1) \log \frac{1}{p(1)},$$

and

$$H^{(2)}(S) = p(00) \log \frac{1}{p(00)} + p(01) \log \frac{1}{p(01)} + p(10) \log \frac{1}{p(10)} + p(11) \log \frac{1}{p(11)}.$$

We examine the input binary sequence A and the output binary sequence C as a result of the RPB encryption. Since the 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 is equal to those of input A, respectively. Hence, by the definition of the first-order entropy, we have $H^{(1)}(A) = H^{(1)}(C)$.

The situation becomes more complicated for the second-order entropy. The number of digrams 00, 01, 10, 11 will vary under the RPB operation due to rotation of bit blocks. The change of digrams after a bit block rotation is illustrated in Fig. A.3.

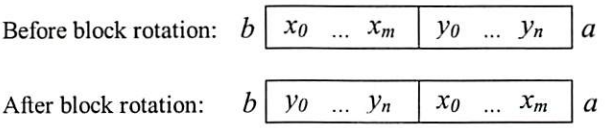


Figure A.3: Change of digrams under the RPB operation.

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

$$b\,x_0 \rightarrow b\,y_0, \quad x_m\,y_0 \rightarrow y_n\,x_0, \quad y_n\,a \rightarrow x_m\,a.$$

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

$$b0 \rightarrow b1, \quad 01 \rightarrow 10, \quad 1a \rightarrow 0a. \tag{A.8}$$

From these digram changes, we see 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

$$b1 \rightarrow b0, \quad 10 \rightarrow 01, \quad 0a \rightarrow 1a,$$

which is clearly the opposite of the change corresponding to $(x_0, x_m, y_0, y_n) = (0, 0, 1, 1)$ as in Eq. (A.8). The digram changes for all 16 possible values of the quadruple (x_0, x_m, y_0, y_n) are given in Table A.1.

$x_0\, x_m\, y_0\, y_n$	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	$00 \rightarrow 10$ $1a \rightarrow 0a$
0 1 0 0	$10 \rightarrow 00$ $0a \rightarrow 1a$
0 0 1 0	$b0 \rightarrow b1$ $01 \rightarrow 00$
1 0 0 0	$b1 \rightarrow b0$ $00 \rightarrow 01$
0 0 1 1	$b0 \rightarrow b1$ $01 \rightarrow 10$ $1a \rightarrow 0a$
1 1 0 0	$b1 \rightarrow b0$ $10 \rightarrow 01$ $0a \rightarrow 1a$
0 1 1 0	$b0 \rightarrow b1$ $11 \rightarrow 00$ $0a \rightarrow 1a$
1 0 0 1	$b1 \rightarrow b0$ $00 \rightarrow 11$ $1a \rightarrow 0a$
0 1 1 1	$b0 \rightarrow b1$ $11 \rightarrow 10$
1 1 0 1	$b1 \rightarrow b0$ $10 \rightarrow 11$
1 0 1 1	$01 \rightarrow 11$ $1a \rightarrow 0a$
1 1 1 0	$11 \rightarrow 01$ $0a \rightarrow 1a$

Table A.1: Counts of digram changes under the RPB operation.

It is clear from Table A.1 that the 16 values of (x_0, x_m, y_0, y_n) can be grouped into 6 pairs with an opposite change effect and 4 values with no effect at all (because $x_0 = y_0$ and $x_m = y_n$). It is reasonable to claim that these 16 values occur with an equal probability for the following reasons.

1. As the result of high performance multimedia compression, input stream A is a nearly random sequence.

2. In the RPB encryption scheme, the partition size and the rotation size are both determined by a pseudo-random sequence. Thus, the partition boundary and the rotation boundary occur uniformly at all possible bit positions.

For a sufficiently long sequence, the rise and fall in the number of digrams 00, 01, 10, 11 will cancel out. Hence, we come to the conclusion that, in a statistical average sense, the RPB operation does not change the counts of digrams in a sequence. By the probability calculation as indicated in Eq. (A.7), the probability of digrams for input A and output C are equal. It follows directly from the definition of the second-order entropy that $H^{(2)}(A) = H^{(2)}(C)$.

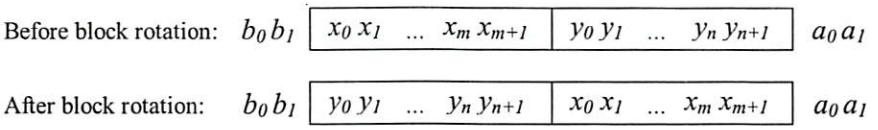


Figure A.4: Change of trigrams under RPB due to the bit block rotation.

We can go one step further to the third-order entropy. The change of trigrams (3-bit subsequence) due to a bit block rotation is shown in Fig. A.4.

Again the bit block $x_0x_1 \ldots x_mx_{m+1}$ is rotated behind bit block $y_0y_1 \ldots y_ny_{n+1}$. b_0b_1 and a_0a_1 are two bits before and after the rotation, either 0 or 1. We can obtain the following trigram changes:

$$b_0\,b_1\,x_0 \rightarrow b_0\,b_1\,y_0,\ b_1\,x_0\,x_1 \rightarrow b_1\,y_0\,y_1,$$

$$x_m\,x_{m+1}\,y_0 \rightarrow y_n\,y_{n+1}\,x_0,\ x_{m+1}\,y_0\,y_1 \rightarrow y_{n+1}\,x_0\,x_1,$$

$$y_n\,y_{n+1}\,a_0 \rightarrow x_m\,x_{m+1}\,a_0,\ y_{n+1}\,a_0\,a_1 \rightarrow x_{m+1}\,a_0\,a_1.$$

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