MEPG Video Encryption Algorithms *

Bharat Bhargava Changgui Shi Sheng-Yih Wang
Department of Computer Science
Purdue University
West Lafayette, IN 47907, USA

August 13, 2002

Abstract. Multimedia data security is important for multimedia commerce. Previous cryptography studies have focused on text data. The encryption algorithms developed to secure text data may not be suitable to multimedia applications because of large data sizes and real time constraint. For multimedia applications, light weight encryption algorithms are attractive.

In this paper, we present four fast MPEG video encryption algorithms. These algorithms use a secret key randomly changing the sign bits of of DCT coefficients and/or the sign bits of motion vectors. The encryption effects are achieved by the IDCT during MPEG video decompression processing. These algorithms add very small overhead to MPEG codec. Software implementations are fast enough to meet the real time requirement of MPEG video applications. Our experimental results show that these algorithms can achieve satisfactory results. We believe that they can be used to secure video-on-demand, video conferencing and video email applications.

Keywords: Multimedia data security, MPEG video encryption, MPEG codec.

1. Introduction

Multimedia data security is a very important issue in both traditional and electronic commerce. For example, in video-on-demand and video conferencing applications, it is imperative that only those who have paid for the services can access the data.

One way to control the access of distributed multimedia data is to use authentication control mechanisms. However, authentication control mechanisms alone are not enough to secure multimedia data which are transmitted on wireless, satellite, or IP networks. The most popular method to secure multimedia data is to treat the whole data as a binary file and encrypt the whole data using secret-key encryption algorithms such as DES (Data Encryption Standard) (Stinson, 1995). Secret key encryptions, however, are very complicated and involve large amount of computations. Software implementations of these schemes are not fast enough to process the vast amount of data generated by multimedia

* E-mail: {bb,shicg,swang}@cs.purdue.edu
This research is supported by CERIAS and NSF grants CCR-9901712 and CCR-0001788.

© 2002 *Kluwer Academic Publishers. Printed in the Netherlands.*

applications and hardware implementations add extra costs to both the data generators and receivers.

The challenges of multimedia data encryption come from two facts. First, the sizes of typical multimedia data are usually very large (for example, the size of a two-hour MPEG-1 video is about 1 GB). Second, multimedia data needs to be processed in real time (for example, the data rates of MPEG-2 can go up to 40Mbps or more). Processing vast amount of data in very short time puts great burden on video codec, storage space requirements and network communications. Heavy-weight encryption and decryption algorithms (during or after encoding phase) will aggravate the problem and increase the latency.

For commercial applications such as pay-per-view video, very expensive attacks of the scrambled multimedia data are not interesting to the adversaries because most multimedia videos are not as valuable as military secrets or financial information (Macq and Quisquater, 1995). For those applications, the information rate is very high, but the information value is very low (Macq and Quisquater, 1995). The cost to break such encryption code is much higher than the cost to buy the programs. Hence, light-weight encryption and decryption algorithms are suitable to secure these kinds of multimedia data.

In this paper, we present four efficient MPEG video encryption algorithms. All these algorithms, although vary in attack-protection strength, add very small overhead to the MPEG codec. Software implementations of these algorithms are fast enough to meet the real time requirement of MPEG video applications. DVD is 4.7 G bytes of removal storage media and can accommodate two and half hours of MPEG-2 encoded motion picture. DVD will expand home video market owing to its clear vision as CD audio conducted a remarkable growth of recorded music sales from. Thus DVD copy protection is becoming new hot research topic now. Since hardware may be used for decoding MPEG bit stream on DVD, developing video encryption technique for hardware environment is also becoming important. The MPEG encryption techniques protect the DVD contents from two possible kinds of pirate devices: DVD video decoders and recordable storage devices. Our technique can be used in encrypting the DVD video decoders.

The rest of this paper is organized as follows. Section 2 is a brief introduction to MPEG-1 compression model. Section 3 discusses related work. Section 4, 5, 6, and 7 describe our four video encryption algorithms. Section 8 shows experimental results. Finally, conclusions and future work are discussed in section 9.

2. Background

In MPEG-1 video coding model (Gall, 1991), a video is composed of a sequence of group of pictures (GOPs). Each GOP is a series of I, P and B pictures. I pictures are intraframe coded without any reference to other pictures. P pictures are predictively coded using a previous I or P picture. B pictures are bidirectionally interpolated from both the previous and following I and/or P pictures. The relative frequency of occurrence of I, P and B pictures can be controlled by the applications.

Each picture is divided into macroblocks. A macroblock is a 16×16 pixel array. Macroblocks belonging to I pictures are spatially encoded. It belonging to B and P pictures are temporally interpolated from the corresponding reference picture(s), and the error between the actual and reference values is computed. The interpolation process also produces up to two motion vectors (a forward prediction vector and a backward prediction vector) for each macroblock in the reference pictures, as illustrated in Figure 2 and Figure 3. Regardless of the type of pictures it belonging to, each macroblock is further subsampled into four 8×8 luminance blocks (Y blocks), two 8×8 chrominance blocks (a Cr block and a Cb block), as shown in Figure 1.

Each of the 8×8 Y, Cb and Cr blocks is fed to a pipeline of DCT (Discrete Cosine Transformation), quantization and Huffman entropy coding, as shown in Figure 1. DCT concentrates most of the energy in the lower spatial frequencies, i.e., the upper-left corner of the 8×8 block. After quantization, many DCT coefficients will become zeros. The quantization output is then linearized according to the zig-zag order to a vector $< DC, AC_1, AC_2, \cdots, AC_{63} >$. DC coefficient denotes the average brightness in the spatial block, and AC coefficients contain detail image information.

The DC coefficients of I blocks are differentially coded. The motion vectors of B blocks and P blocks are also differentially coded.

The run length encoding (RLE) turns vector $< AC_1, AC_2, \cdots, AC_{63} >$ into a sequence of (skip, value) pairs. Then the Huffman entropy coding is used to change the (skip, value) pair sequence into a compressed bitstream. MPEG standard provides a Huffman codeword table, where more frequently occurring values are assigned fewer bits in order to achieve high compression rate. Every Huffman codeword reserves a sign bit, 0 mean positive, 1 mean negative. These **sign bits** are the positions we encrypt the video in our RVEA algorithms in section 7.

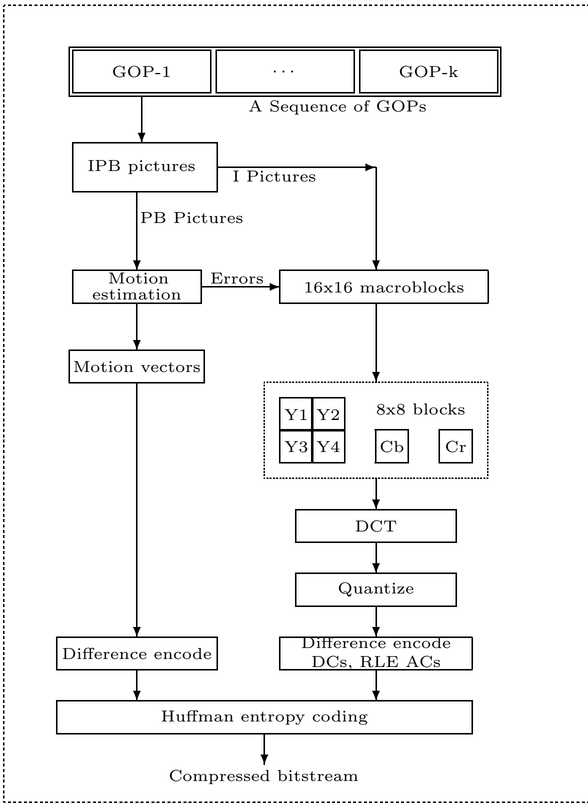

Figure 1. MPEG-1 video coding.

3. Related Work

In many multimedia data scrambling systems such as *vic* (McCanne and Jacobson, 1995) and (Maples and Spanos, 1995), data are treated as bitstreams and encrypted with DES (Stinson, 1995). However, software implementations of DES are too slow to meet the real-time requirement of MPEG video playback. One strategy to solve this problem is to selectively encrypt only portions of MPEG video (Li et al., 1996; Maples and Spanos, 1995).

One selection scheme proposed in the literature is to encrypt only MPEG video headers (including GOP headers, slice headers and macroblock headers). However, this scheme is not effective against attack due to the following reasons. First, the headers contains mostly standard information. Attackers can easily guess the information. Second, in many MPEG applications, a video stream is indexed by frame in order to perform synchronization. Therefore, the beginning of each frame

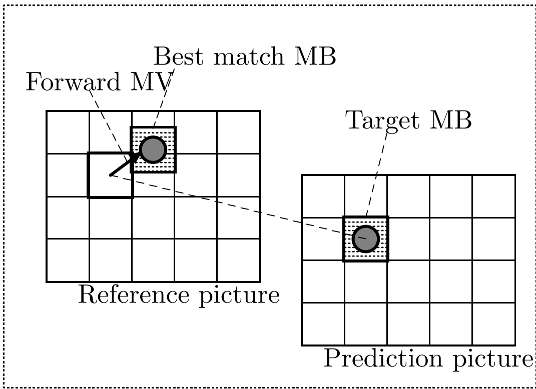

Figure 2. Motion-compensated prediction

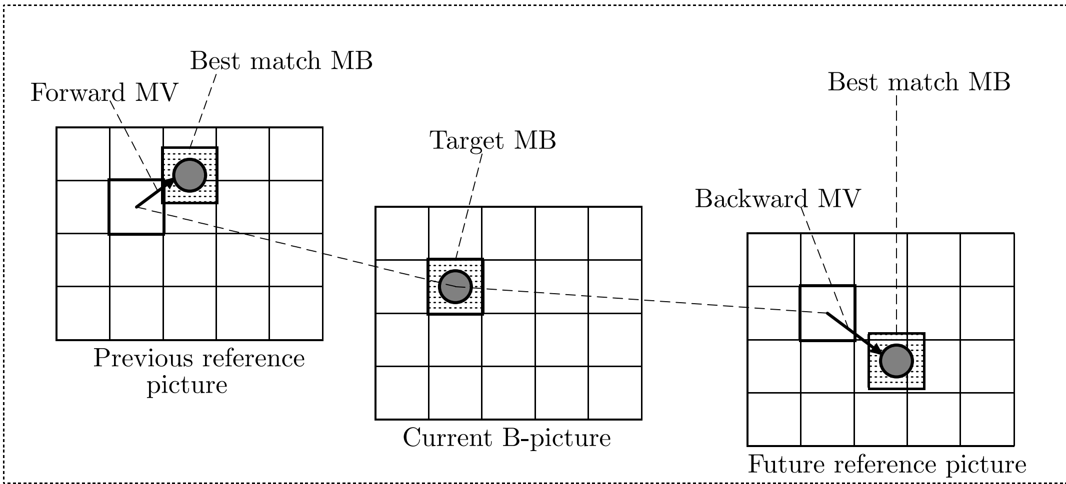

Figure 3. Motion-compensated Interpolation.

is known (Qiao and Nahrstedt, 1998). Third, when block encrypted video headers are sent over a noisy channel, bit errors occurred in an encrypted block will propagate to the whole block when decrypting the video. It can make a decoder lose synchronization during a video playback.

Another scheme to selectively encrypt the video is to encrypt I frames only. One may think that P and B frames are useless without knowing the corresponding reference I frames. But a great portion of the video could be visible without I frames because some of the P and B frames may contain intra-coded I blocks. Therefore, encrypting only

I frames does not provide a satisfactory security level (Agi and Gong, 1996).

Other selective MPEG video encryption methods include (1) encrypting all headers plus DCs and lower AC terms (Meyer and Gadegast, 1995); (2) encrypting one half of a frame with DES/IDEA, and the other half of the frame with "one-time-pad" generated from that frame (Qiao and Nahrstedt, 1998).

In those systems, the encryption operations are processed after MPEG compression operations; the decryption operations are processed before MPEG decompression operations. Those systems add latency to real-time video delivering and overhead to video decoding.

Tang (Tang, 1996) introduced some methods to incorporate MPEG compression with encryption in one step. Tang's methods use a random permutation list to replace the zig-zag order to map the DCT coefficients to a 1×64 vector. Since mapping according to the zig-zag order and mapping according to a random permutation list (the secret key) have the same computational complexity, the encryption and decryption add very little overhead to the video compression and decompression processes. However, Tang's methods decrease the video compression rate. The reason is that the random permutations distort the probability distribution of DCT coefficients and render the Huffman table used less than optimal.

4. Algorithm I

Algorithm I (Shi and Bhargava, 1998c) is a light-weight MPEG video encryption algorithm which incorporates encryption (decryption) with MPEG video compression (decompression) in one step. The primary goal of algorithm I is to save the encryption computation time by taking the advantage of combining MPEG compression and data encryption and at the same time avoid decreasing the video compression rate (Shi and Bhargava, 1998c). We uses a permutation of the Huffman codeword list as a secret key. During MPEG encoding (decoding), our encoder (decoder) uses the secret key instead of the standard Huffman codeword list. For those who do not have the secret key and use the standard Huffman codeword list to decode an encrypted MPEG video, the decoded video frames will be different from the original video frames.

In order to prevent the encryption from affecting compression rate, we developed a algorithm as follow:

Let H be the Huffman codeword list provided by MPEG-1 standard. We use H', a permutation of H, as secret key. The permutation (key generation), $f : H \xrightarrow{1:1} H'$, has the following properties:

1. $\mid w \mid = \mid f(w) \mid$ for $\forall w \in H$, where $\mid w \mid$ is the length (number of bits) of codeword w. This guarantees that H', the permutation of Huffman codeword list, does not change video compression rate.

2. $\|H - H'\| > \beta$, where $\| \cdots \|$ is the L_2 distance defined as

$$\|H - H'\| = \sqrt{\sum_{i=0}^{n} (H[i] - H'[i])^2}$$

$\|H - H'\| > \beta$ guarantees encryption quality. β is a sufficient large number which is selected by users.

The MPEG video data encryption and decryption are embedded in compression and decompression respectively.

– Encryption: during MPEG-1 video compression processing, we use H' to replace H.

– Decryption: during MPEG-1 video decompression, for $\forall w' \in H'$, we find $f^{-1}(w')$ and use it as the real Huffman codeword value.

Algorithm I has both an advantage and a disadvantage:

– The advantage is that no overhead is added to MPEG codec since the encryption/decryption processes does not cost extra computation.

– The disadvantage is the size limitation of the key space. First, since MPEG compression rate depends on the Huffman codeword list, if we use an arbitrarily Huffman codeword list to encode the MPEG video, the compression rate may decrease. To avoid affecting compression rate, we limit the permutation of Huffman codeword list (the secret key) to those codewords which have the same length as the standard Huffman codeword. Second, it seems that not all of permutations of the Huffman codeword list can be used as encryption keys. This makes key generation difficult since a generated key has to be tested for validity before using.

5. Algorithm II — VEA

To remedy the drawbacks of Algorithm I, we developed a video encryption algorithm called VEA (Shi and Bhargava, 1998b) which is described in this section.

Before describing the details of VEA, it is worthwhile to describe the general scheme of multimedia data encryption. Multimedia data encryption techniques aim to prevent unauthorized receivers from decoding the data by scrambling them. The basic principle is to apply an invertible transformation E_{k_1} to the original data (or a portion of them) S (called *plaintext*) and produces a bitstream C (called *ciphertext*) such that $C = E_{k_1}(S)$. An authorized receiver who has a secret key, k_2 can decrypt the data by the transformation $D_{k_2} = E_{k_1}^{-1}$. The decryption operation is

$$D_{k_2}(C) = E_{k_1}^{-1}(C) = E_{k_1}^{-1}(E_{k_1}(S)) = S$$

where the parameter k_1 is called the encryption key and k_2 is called the decryption key.

Previous encryption algorithms have focused on text data, and they may not be suitable to multimedia applications because of the nature of multimedia data (large size, low value and real-time constraint). Security is a trade-off between the cost of the data being protected and the cost attackers pay to get that data. The costs of a multimedia security system include the amount of investment by data provider and the payment required for customer service. In multimedia applications, data providers such as VOD providers do not want to spend too much money on their security equipments. Data receivers do not want to buy extra devices (hardware boxes or co-processors) just to decode the data. In this case, encryption algorithms which require less computation and low implementation cost become very attractive. For multimedia data, we only need to provide sufficient security level such that the cost to break the security system is much more expensive than to buy the data. Therefore, light-weight and cost-effective multimedia data encryption algorithms should be developed.

5.1. The Algorithm

Due to the special property of MPEG video structure, it is not necessary to encrypt the video bit by bit. Our MPEG video encryption algorithm, VEA, is a selective encryption algorithm which only operates on the sign bits of DCT coefficients of a MPEG compressed video. VEA's secret key, k, is a randomly generated bitstream of length m which can be represented as $k = b_1 b_2 \cdots b_m$. A MPEG compressed video S is a bitstream which can be represented as

$$S = \cdots s_1 \cdots s_2 \cdots s_m \cdots s_{m+1} \cdots s_{m+2} \cdots s_{2m} \cdots$$

where s_i $(i = 1, 2, \cdots)$ are all of the sign bits of DC and AC coefficients (for DC coefficients of Y, Cr and Cb blocks of I-frames, these are the sign bits of the differential values, since DC coefficients are differentially coded).

VEA's encryption function E_k can be described as

$$E_k(S) = \cdots (b_1 \oplus s_1) \cdots (b_m \oplus s_m) \cdots (b_1 \oplus s_{m+1}) \cdots (b_m \oplus s_{2m}) \cdots$$

where $\oplus$ is the binary XOR operation. The encryption operations randomly change the sign bits of DCT coefficients. According to the given secret key, a sign bit is either not changed (if the corresponding bit of the key is 0), or changed from positive to negative (0 to 1), or changed from negative to positive (1 to 0).

Pseudocode in Table I describes VEA encryption operations in detail. (Table I should not be considered as an optimal implementation of VEA, it only explains the algorithm). VEA takes a secret key and an MPEG video file as the input parameters. Its output is a VEA encrypted MPEG video file.

In Table I, we add resynchronization points at the beginning of each GOP(Group Of Picture). Resynchronization points are the positions in MPEG video data stream where VEA starts to reuse its secret key from the first bit. This feature is useful because video transmission in networks is not reliable. Noise and errors exist in most transmission media, leading to lost data bits or even lost video frames (Bhargava et al., 1996a; Bhargava et al., 1996b; Bhargava et al., 1996c; Zeng and Liu, 1995). In addition, VCR-like functions such as fast forward, rewind of data streams (Little and Venkatesh, 1994) in VOD systems also require that VEA can start to decrypt video at many points. Resynchronization points allow users to decrypt only certain segments of a video. In a VEA implementation, one can decide his/her own resynchronization points. Some possible candidates for resynchronization points are the beginning of each I frame, the beginning of each GOP, or the beginning of every 30th frame.

5.2. Analysis

VEA's encryption effects are achieved by the IDCT process during MPEG video decoding. Even if only some of DCT coefficients are changed, these changes will propagate to most of IDCT coefficients.

For those who have the secret key, they can decrypt the received data and get the original video. VEA's decryption function, E_k^{-1} is the same as its encryption function since $E_k(E_k(S)) = S$. In VEA, encryption key and decryption key are the same. For those who do not know the secret key, their decoders will produce images very different from the original video because most image pixel values are changed.

Table I. VEA Algorithm

```
Algorithm II: VEA(
  int m,             /* key length */
  bit key[m],        /* secret key */
  char *mpeg_video,    /* input file */
  char *vea_mpeg_video)    /* output file */
{
  int n;             /* buffer size */
  bit video[n];      /* input buffer */
  file in;           /* for input */
  file out;          /* for output */
  int k,l,i=0;
  in=open(mpeg_video,"r");
  out=open(vea_mpeg_video,"w");
  while(!eof(in)) {
    l=read(video, n, in);    /* read l bits */
    for(k=0;k<l; k++){
       switch(video[k]) {
         case (beginning of a GOP):
           i=0; /* resynchronization */
           break;
         case (a sign bit of DCT coefficient
           or a sign bit of DC differential
           value of Y, Cr or Cb block
              of I frame):
           video[k]=video[k] xor key[i];
           i= ++i mod m;
       } /* end switch */
    } /* end for */
    write(video, l, out);    /* write l bits */
  } /* end while */
  close(in); close(out);
} /* end procedure */
```

To understand the effect of error propagation for IDCT coefficients, suppose X is a 8×8 image block. After DCT transformation, we get a 8×8 Y block, $Y = DCT(X)$. When we apply VEA encryption operation to Y (for simplicity, we ignore the MPEG quantization, the zig-zag order and the Huffman entropy coding operations), we actually change the elements of Y according to the given secret key. The encryption results, Z, can be represented as $Z = VEA(Y)$. A decoder (after decryption) can get an image block $\tilde{X}$,

$$\tilde{X} = IDCT(VEA^{-1}(Z)) = IDCT(Y).$$

$\tilde{X}$ is close to but not exactly the same as the original image pixel value X because of the lossy property of MPEG compression processes.

If one does not decrypt Z before the IDCT computations of MPEG decoding (again for simplicity, we ignore the MPEG dequantization, the reverse zig-zag order and the Huffman entropy decoding operations), then his decoder will get image block $\hat{X}$,

$$\hat{X} = IDCT(Z) = \tilde{X} + N$$

where N is the noise or shift added to pixel X. For 8x8 matrix A, if we define

$$|A| = \sum_{i=0}^{7} \sum_{j=0}^{7} |a_{ij}|$$

then

$$|\hat{X} - \tilde{X}| = |N|.$$

The value of $|N|$ will vary with X and the secret key used by VEA.

The analysis above shows that although VEA only changes some signs of DCT coefficients, most image pixel values will be changed during inverse DCT (IDCT) of MPEG decoding operations. The analysis also shows that VEA actually is an image cipher algorithm. It shifts image pixel values according to a given secret key.

For a MPEG compressed video, if it has n non-zero DCT coefficients after quantization, then the computational complexity of VEA is $O(n)$ since it only does one XOR operation for each non-zero DCT coefficient. The number n is dependent on the original MPEG video. To get a rough idea of what percentage the sign bits are to the original video data, we perform a statistic on a segment of the "table tennis" video. The aggregated size of sign bits in DCT coefficients is only about 13.8% percent of the original video bitstream. Since VEA only operates on a small portion of the original video stream, it is fast comparing to those algorithms which operate on the whole video bitstream.

VEA is much more efficient than DES algorithm because it only selectively encrypts a small number of bits of the MPEG compressed video, and a selected bit is only XORed one time with the corresponding bit of the secret key. In contrast, DES encrypts every bit of the compressed video, and each bit is XORed 8 times with the given secret key plus many transposition operations. To encrypt a MPEG video, a software implementation of DES will take much more time than a software implementation of VEA does.

To attack a VEA encrypted MPEG video, an adversary needs to try, on the average, 2^{m-1} times for a secret key of length m to determine the real secret key. For example, if $m = 56$, the size of the VEA key space is the same as the size of the DES key space. It is very difficult to break an encryption algorithm with this size of key space using currently known techniques. In addition, the cost of determining the secret key is increased by MPEG decoding process which includes inverse DCT computations.

VEA has following properties:

1. One can encrypt a MPEG video many times, and decrypt the video in one step. The reason is that for any two keys k_1 and k_2 ($k_1 \neq k_2$), VEA has a property of

   $$E_{k_1}(E_{k_2}(S)) = E_{(k_1 \oplus k_2)}(S)$$

   If we define an operation k_3 to be $k_3 = k_1 \oplus k_2$, then one can use k_3 to decrypt the encrypted video in one step.

   This property is very useful when the secret keys need to be changed periodically to prevent adversary attack. It can be derived from this property that the cost of changing the secret key in VEA is cheaper than the cost of changing the secret key in DES, which has to perform the decryption first and than perform the encryption again.

2. In VEA, the length of a secret key is not limited. Using a longer secret key will not increase the complexity of encryption computations, but it will make adversaries more difficult to attack.

   In general, the *block encryption* algorithms developed to secure text data usually use short keys and complex computations. This is because that the size of text data are small and cryptography experts want to make it computationally impossible for adversaries to determine the secret key even with the help of the most powerful computers. For multimedia applications, we believe that the reverse is also true in many cases. We can use longer keys to prevent adversary attack and use simple computation algorithms to ensure real time performance. Since multimedia data usually has a larger size, the adversary needs to spend a much longer time to perform the computations in order to determine the real secret key.

Of course, using longer keys makes key distribution difficult. One solution is to use a pseudo-random number generator. The key distributed is used as the seed of the random number generator, and the key actually used is generated by the generator. Thus, we turn our algorithm into a selective *stream encryption* algorithm. One can use an arbitrarily long key.

VEA's advantages and disadvantages:

- Due to the well know MPEG structure and DCT coefficient distributions, a cryptanalysis expert may obtain obscure images by using a partial key. Therefore VEA is not ideal for highly valuable multimedia applications. However, it is robust for those multimedia applications where the cost to break the encryption MPEG video is much higher than that to buy the video. VEA can be used to secure pay-per-view video and MPEG video lectures, or to prevent children from watching adults movies, etc.

- VEA, like Tang's algorithm (Tang, 1996), is weak for plaintext attack. If an adversary knows the original video image (the *plaintext*) and the encrypted video (the *ciphertext*) (for example, by subscribing to the program) he can easily determine the VEA secret key. Therefore it is worthwhile to increase the VEA security level by frequently changing its secret key.

- When applied to MPEG video delivery, some algorithms encrypt every bit of the video bitstream, including the long run of zeros for resynchronization and the MPEG header bits. These encrypted long run of zeros and encrypted MPEG header bits can be used to help the adversaries determine the secret key (Macq and Quisquater, 1995). VEA does not have this drawback since it does not encrypt the synchronization bits and MPEG header bits. Of course, deliberated VEA users will not encrypt the advertisement part of their videos since those advertisements (the plaintext) may appear in other video programs which can be easily obtained by the adversaries.

- Since VEA does not change the MPEG video structure, a MPEG video encrypted with VEA is still readable to MPEG tools, while a MPEG video encrypted with DES is not readable to common

MPEG tools. Hence, our encryption algorithm is weaker than DES in this aspect.

6. Algorithm III – MVEA

6.1. The Algorithm

One improvement over the VEA algorithm in the previous section is: in addition to randomly changing the sign bits of differential values of DC coefficients of I frames (I block), the sign bits of differential values of motion vectors in B and P frames can also be randomly changed. A modified version of VEA algorithm, called MVEA (Shi and Bhargava, 1998a), is described in Table II.

MVEA takes a secret key and a MPEG video file as input parameters. Its output is a MVEA encrypted MPEG video file. Similar to VEA algorithm, we add resynchronization points at the beginning of each GOP (Group Of Picture). Other choices for resynchronization points is also possible as discussed in previous section.

Comparing to the VEA algorithm, MVEA includes motion vectors in the video encryption. We found that encrypting motion vectors is very efficient. It makes encryption of DCT coefficients of B and P frames unnecessary. Because motion vectors are differentially encoded, changes in the sign bits of the differential values of motion vectors not only lead to changes in the directions of motion vectors but also change the magnitudes of motion vectors.

In MVEA, we only encrypt DC coefficients of I frames, and leave AC coefficients of I frames unchanged. We, thus, significantly reduce encryption computations. Because DC coefficients of I frames are differentially encoded, changing a few sign bits of differential values of DC coefficients will affect many DC coefficients during MPEG decoding. MPEG's differential code of DC coefficients and motion vectors increases the difficulty to break MVEA encrypted videos.

Because DC coefficients and AC coefficients are related, some adversaries may use AC coefficients to derive DC coefficients because the sum of AC coefficients for each block is the times of its DC coefficient (see appendix). Hence, for high value MPEG video, we still recommend users changing the sign bits of AC coefficients of DCT.

6.2. Analysis

Changing the sign bits of motion vectors of P and B frames will mislead the adversary's decoder during video playback. When we change the

Table II. MVEA Algorithm

```
Algorithm III: MVEA(
  int m,            /* key length */
  bit key[m],       /* secret key */
  char *mpeg_video,    /* input file */
  char *vea_mpeg_video)    /* output file */
{
  int n;            /* buffer size */
  bit video[n];     /* input buffer */
  file in;          /* for input */
  file out;         /* for output */
  int k,l,i=0;
  in=open(mpeg_video,"r");
  out=open(vea_mpeg_video,"w");
  while(!eof(in)) {
    l=read(video, n, in);    /* read l bits */
    for(k=0;k<l; k++){
        switch(video[k]) {
          case (beginning of a GOP):
          i=0; /* resynchronization */
          break;
        case (sign bit of DC differential value
          of Y, Cr or Cb block of I frame):
          video[k]=video[k] xor key[i];
          i= ++i mod m;
          break;
        case (sign bit of differential value
          of motion vectors of B or P frame):
          video[k]=video[k] xor key[i];
          i= ++i mod m;
        } /* end switch */
    } /* end for */
  write(video, l, out);    /* write l bits */
  } /* end while */
  close(in); close(out);
} /* end procedure */
```

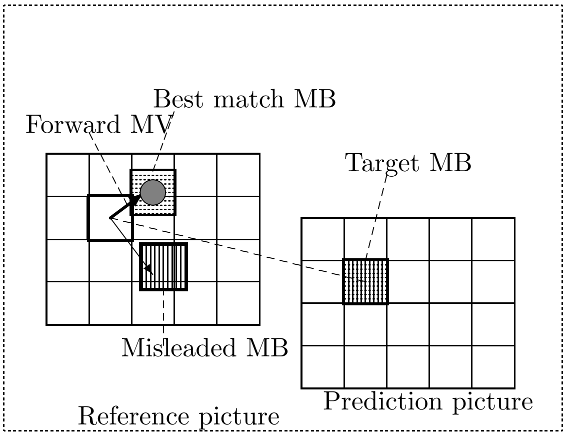

Figure 4. RVEA misleads adversaries' decoders to refer wrong macroblock

sign bits of differential values of motion vectors, we actually change the directions of motion vectors. Because motion vectors are differentially encoded, the changes will also affect the magnitudes of motion vectors during decoding. Figure 4 illustrates the scenario for a P frame. Similar figure can be drawn for B frames. When decoding P and B frames, RVEA misleads the adversary's decoder to refer to wrong reference macroblocks. Hence, the adversary's decoder will display pictures different from the original video frames.

7. Algorithm IV — RVEA

In our VEA and MVEA algorithms, we use a secret key to randomly change the sign bits of all DCT coefficients and the sign bits of motion vectors. These algorithms can achieve satisfying encryption results with less computations. However, they are very weak in plaintext attack. If an adversary has several frames of plaintext and the whole ciphertext, he can easily find secret key and decrypt the whole video. In this section, we present an algorithm called RVEA (Shi et al., 1999) which is robust under both plaintext and ciphertext attacks.

RVEA is a selective encryption algorithm which operates only on the sign bits of both DCT coefficients and motion vectors of a MPEG compressed video. RVEA can use any secret key cryptography algorithms (such as DES or IDEA) to encrypt those selected sign bits.

In order to illustrate the operations of RVEA, if we ignore the other bits and concentrate only on the sign bits of an MPEG video stream, then an MPEG video S can be represented as

$$S = \cdots s_1 \cdots s_2 \cdots s_m \cdots$$

where s_i $(i = 1, 2, \cdots)$ are the sign bits of motion vectors or the sign bits of DCT coefficients (for DC coefficients of I frames and motion vectors, these sign bits are the sign bits of the differential values, since they are differentially coded).

RVEA's encryption function, E_k (E is DES or IDEA), encrypts MPEG video slice by slice, as shown in Table III. RVEA randomly changes the selected sign bits of DCT coefficients and sign bits of motion vectors. According to the given secret key, a sign is either not changed, or changed to its opposite sign.

RVEA selects at most 64 sign bits (8 bytes) from each macroblock. Let us denote an MPEG compressed 8×8 block as a bitstream of $\beta\alpha_1\alpha_2\alpha_3 \ldots \alpha_n$ where β is the code of DC coefficient (if this block belongs to an I frame, then β is the code of the differential value, since the DC components of I frames are differentially encoded), α_i is the code for ith nonzero AC coefficient.

Each 16×16 macroblock contains six 8×8 blocks. They are Y1, Y2, Y3, Y4, Cr, and Cb. Figure 5 defines the order which RVEA selects the sign bits of β and α_i from each macroblock. The reason to select sign bits in this way is that DC coefficients are more significant than AC coefficients and lower frequency ACs are more significant than higher frequency ACs. RVEA achieve the goal of reducing and bounding its computation time by limiting the maximum number of bits selected.

RVEA's decryption operations are the same as its encryption operations.

Changing the sign bits of DC coefficients of I frames can lead to a very dramatic change in the decoded frame. Since DC coefficients of I frames are differentially encoded, changing the sign bit of a differential value will affect all of its following DC coefficients during MPEG decoding. As an example, Figure 6(a) is a plot of the sequence of DC values of a I frame. Figure 6(b) shows the differential values of those DC values. RVEA encryption randomly changes the signs of these encoded values, as shown in Figure 6(c). Without a correct secret key, a decoder will get wrong DC values, as shown in Figure 6(d).

The differential encoding of DC coefficients and motion vectors in MPEG compression increases the difficulty of breaking RVEA encrypted videos. If the initial guess of a DC coefficient is wrong, it is very difficult to guess the following DC values correctly.

Theoretically, the difficulty of breaking an RVEA encrypted MPEG video is the same as the difficulty of breaking the underlining secret key cryptography algorithms. Even if plaintext and ciphertext are known,

Table III. RVEA Algorithm

```
Algorithm IV: RVEA:
{
  for (each macroblock M in a video slice){
    switch(M) {
        case (belong to a I frame):
            select at most 64 DC and AC
            sign bits in the order
            given in Figure 5;
            break;
        case (belong to a P frame):
            first select the sign bits of forward
            motion vector (f_x, f_y);
            then select at most 62 DC and AC
            sign bits in the order
            given in Figure 5;
            break;
        case (belong to a B frame):
            first select the sign bits of forward
            motion vector (f_x, f_y) and backward
            motion vector (b_x, b_y);
            then select at most 60 DC and AC
            sign bits in the order
            given in Figure 5;
        } /* end switch */
  } /* end for */
  encrypt the selected bits with DES or IDEA;
  put the encrypted bits to their
  original positions;
} /* end function */
```

currently there is no practical method to determine the secret key in DES or IDEA. Please note that the cost of plaintext attack to RVEA is increased by the MPEG decoding process, which includes expensive IDCT computations.

Ciphertext attacks on RVEA encrypted MPEG videos are not practical either. Ciphertext attacks on RVEA can be done by trying all

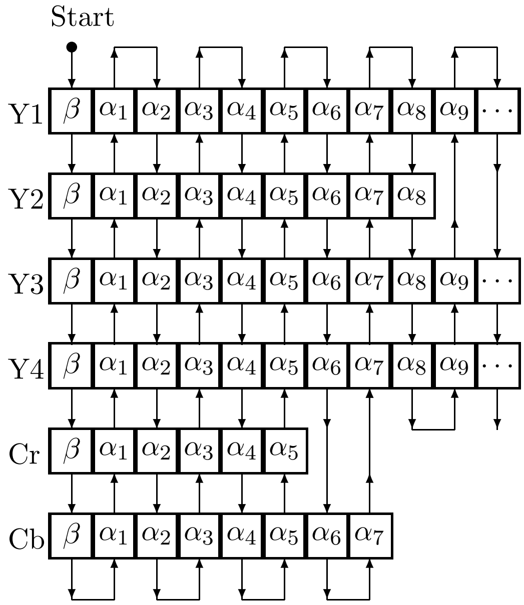

Figure 5. Bits selection order

possible combinations of the sign bits of a frame and check if any one of the combination of the sign bits can produce a comprehensible video frame. The decorrelative property of AC coefficients of DCT makes a correlation attack not very easy. Even with a powerful computer, an automatic ciphertext attack (revealing secret key or generating quality pictures without human involvement) is hard, because the computer does not know if a guessed picture is comprehensible to human beings. A ciphertext attack with human involvement by playing picture puzzles of flipping the sign bits may get some blurred pictures or figure out some objects in certain frames. It will take many hours for an attacker to get a blurred picture. Anyone who found the picture interesting would rather buy the video than attack the ciphered video because the cost of buying the program is actually cheaper.

RVEA only selectively encrypts a fraction of the whole video. It is faster than encrypting the whole video with DES/IDEA. We found that in typical MPEG videos sign bits occupy less than 10% of a whole video bitstream. Therefore RVEA can save 90% of encryption time comparing to algorithms which encrypt the whole video.

RVEA encrypts at most 64 bits (8 bytes) for each macroblock. Thus, it considerably reduces encryption computations. For example, for 320 $\times$ 240 video frames, there are 20 $\times$ 15 macroblocks in each

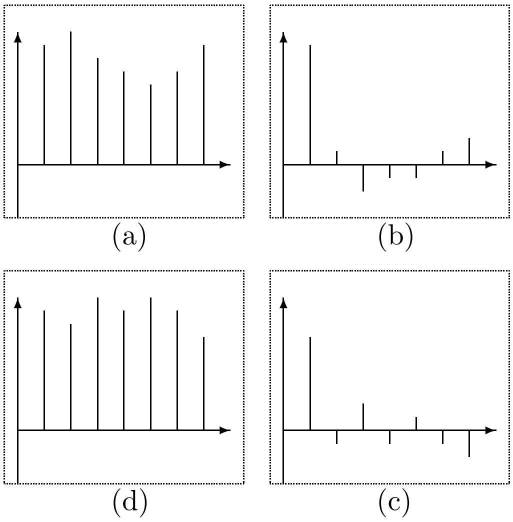

Figure 6. Encryption of differential values of DCs of I frames. (a) Signal values; (b) Differential values; (c) Encrypted differential values; (d) Result of decoding without decryption.

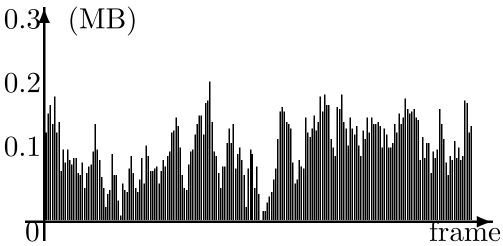

Figure 7. Example of video frame sizes.

frame. To process 30 frames per second, RVEA needs to encrypt data at $20 \times 15 \times 30 \times 8$ bytes per second, which is about 72 Kb/sec. Using IDEA, even a 66Mhz 486 PC can encrypt data at 300kb/sec (Schneier, 1996).

We do realize that applying RVEA encryption to the unit of video slice implies that the decryption can start only after the full slice is available; and the decryption followed by sign correction may impose certain amount of delay. But because the time spent in encryption computations is much longer than the communication delay, so RVEA is still much faster than the approaches which encrypt the whole video frames.

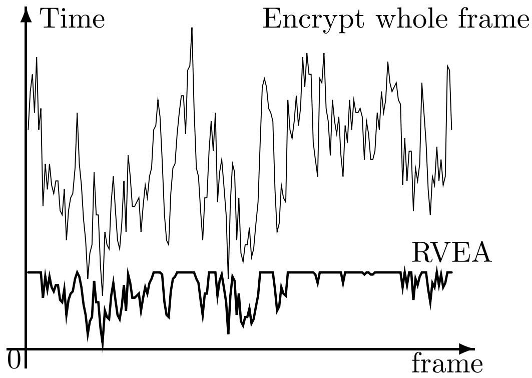

Figure 8. RVEA encryption time for each frame is bounded.

To guarantee QoS (Quality of Service) in real-time video applications, it is desirable that the encryption/decryption time be bounded by a constant. Encryption and decryption should not take too much time, otherwise the video session will suffer from jitters. This is especially true for MPEG video applications because MPEG video frame sizes vary in time, as shown in Figure 7.

The encryption time spent by the algorithms which encrypt whole frames may vary with frame sizes. Larger frame size requires longer encryption time, which means longer delay. It may either degrade the video playback or add extra burden to the video frame synchronization.

Burstiness of data does not affect RVEA's encryption speed. RVEA encrypts at most 64 bits (8 bytes) of data for each macroblock, no matter how large the frame size is and what type (I, P, or B) the frame is (see Figure 8). This is a very desirable property to video decoding.

8. Experiments

We conducted our experiments on a Sun Sparc 10 station at the Raid Laboratory of the Computer Science Department of Purdue University. We modified the Berkeley *mpeg_play* (Patel et al., 1993) and *mpeg_encode* (Gong and Rowe, 1994) programs and used them to test our algorithms.

There are two purposes in our experiments:

1. to test the encryption results;

2. to find the overheads added to MPEG codec.

We selected key length of 128 bits. To test RVEA encryption results, we used IDEA in our RVEA implementation because IDEA is faster than DES. IDEA uses a key of 128 bits to encrypt a plaintext block of 64 bits. IDEA is generally considered to be very secure. No practical attack on it has been published.

Our experimental results:

- The input data is the “table tennis” MPEG-1 video clips. One frame of the original clip is shown in Figure 9.
- Encrypting all AC coefficients of DCT with VEA. One frame of the results is shown in Figure 10. The video image is blurred, but still comprehensible.
- Encrypting all AC coefficients and DC coefficients of Cr and Cb blocks with VEA. One frame of the results is shown in Figure 11. The image is obscured, but still comprehensible.
- Encrypting all DC and AC coefficients, including DCs of Y blocks with VEA. One frame of the results is shown in Figure 12. The image is incomprehensible.
- Figure 13 shows the effect of encrypting only motion vectors of a B frame with MVEA.
- Figure 14 shows one of the RVEA encrypted frames. All pictures in our RVEA tests are incomprehensible.

¿From our experiments, we found that motion vectors are more significant than other DCT coefficients, and DC coefficients are more significant than the AC coefficients in an encryption. When we encrypted all DC coefficients of Y blocks, we always got satisfactory encryption results.

We embedded VEA into the Berkeley *mpeg_encode*(Gong and Rowe, 1994) program and used it to test the overheads added to MPEG codec. We conducted 10 tests each with different keys and encrypted all of DC and AC coefficients. All of the tests got incomprehensible video images. The time spent on compression only and compression with encryption is shown in Table IV (where **fps** means frame per second).

¿From Table IV we can see that relative time spent on the encryption is very small, only 1.81% of computation time. Hence, our software implementation of VEA is fast enough to secure MPEG video applications.

We found that the time spent on encryption with RVEA during the video compression process is only 2.55% of the total computation time. Hence, a software implementation of RVEA is fast enough to secure MPEG video in real-time MPEG applications.

Table IV. Overheads test results

Test number	compression without encryption (fps)	compression and encryption (fps)
1	0.661813	0.672043
2	0.656599	0.657030
3	0.652742	0.678426
4	0.665779	0.692521
5	0.651042	0.659631
6	0.662691	0.665890
7	0.679348	0.679841
8	0.651042	0.673854
9	0.660939	0.665466
10	0.659631	0.678929
Average	0.660163	0.672363

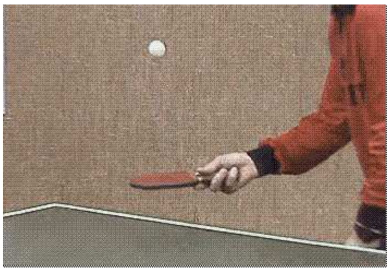

Figure 9. Original image frame.

Figure 10. Encrypted AC coefficients with VEA

Figure 11. Encrypted ACs, DCs of Cr and Cb blocks with VEA

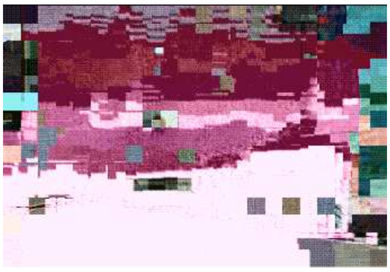

Figure 12. Encrypted all ACs and DCs with VEA

Figure 13. Encrypted motion vectors only with MVEA

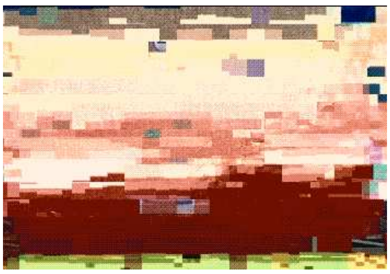

Figure 14. A RVEA encrypted frame

9. Conclusions

We have developed and tested four MPEG video encryption algorithms. These algorithms can achieve satisfactory encryption results with less computations. Software implementations are fast enough to meet the real-time requirement of MPEG decoding. We believe that these algorithms can be used to secure video-on-demand applications and pay-per-view programs. It is also possible to extend the ideas of these four algorithm to non-MPEG compression scheme such as **H.263** for video-conferencing applications.

References

Agi, I. and L. Gong: 1996, 'An Empirical Study of MPEG Video Transmission.'. *In Proceedings of the Internet Society Symposium on Network and Distributed Systems Security, pp137-144. San Diego, CA.*

Bhargava, B., S. Li, S. Goel, M. Annamalai, P. Tang, and Y. Zhang: 1996a, 'Impacts of Codec Schemes on Multimedia Communications'. In: *Proceedings of the International Conference on Multimedia Information Systems (MULTIMEDIA 96), New Delhi, India*. pp. 94–105.

Bhargava, B., S. Li, S. Goel, and J. Huai: 1996b, 'A Distributed Video-on-Demand System for Video Conferencing'. In: *Proceedings of the International Conference on Multimedia Information Systems (MULTIMEDIA 96), New Delhi, India*. pp. 83–93.

Bhargava, B., S. Li, S. Goel, C. Xie, and C. Xu: 1996c, 'Performance Studies for an Adaptable Video-Conferencing System'. In: *Proceedings of the International Conference on Multimedia Information Systems (MULTIMEDIA 96), New Delhi, India*. pp. 106–116.

Gall, D. L.: 1991, 'MPEG: A Video Compression Standard for Multimedia Applications'. *Communications of the ACM, 34(4):46-58*.

Gong, K. L. and L. A. Rowe: 1994, 'Parallel MPEG-1 Video Encoding'. *In Proceedings of the 1994 Picture Coding Symposium*.

Li, Y., Z. Chen, S. Tan, and R. Campbell: 1996, 'Security Enhanced MPEG Player.'. *In proceedings of IEEE First International Workshop on Multimedia Software Development (MMSD 96), Berlin, Germany*.

Little, T. and D. Venkatesh: 1994, 'Prospects for Interactive Video-on-Demand'. *IEEE, Multimedia 1(3):14-24*.

Macq, B. and J. Quisquater: 1995, 'Cryptoloy for Digital TV Broadcasting'. *Proceedings of the IEEE, Vol. 83(6):944-957*.

Maples, T. B. and G. A. Spanos: 1995, 'Performance Study of a Selective Encryption Scheme for the Security of Networked, Real-time Video'. *In Proceedings of the 4th International Conference on Computer Communications and Neworks*.

McCanne, S. and V. Jacobson: 1995, '*vic*: A Flexible Framework for Packet Video'. *In Proc. of the ACM Multimedia'95, pages 511-522, San Francisco, California*.

Meyer, J. and F. Gadegast: 1995, 'Security Mechanism of Multimedia Data with the Example MPEG-1 Video.'. *Available on WWW via http://www.powerweb.de/phade/phade.html*.

Patel, K., B. Smith, and L. Rowe: 1993, 'Performance of a Software MPEG Video Decoder'. *In Proceedings of ACM Multimedia 93.*

Qiao, L. and K. Nahrstedt: 1998, 'Comparison of MPEG Encryption Algorithms'. International Journal on Computers&Graphics*, Special Issue: "Data Security in Image Communication and Network", Vol. 22, No. 3, published bimonthly by Permagon Publisher.*

Schneier, B.: 1996, *Applied Cryptography.* New York: John Wiley & sons, INC.

Shi, C. and B. Bhargava: 1998a, 'An Efficient MPEG Video Encryption Algorithm'. *In Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems, West Lafayette, Indiana, USA. published by IEEE Computer Society, pp381-386. Los Alamitos, California.*

Shi, C. and B. Bhargava: 1998b, 'A Fast MPEG Video Encryption Algorithm'. *In Proceedings of the 6th ACM International Multimedia Conference, Bristol, UK* pp. 81–88.

Shi, C. and B. Bhargava: 1998c, 'Light-weight MPEG Video Encryption Algorithm'. In: *Proc. of the Int'l Conf. on Multimedia (Multimedia98, Shaping the Future), January 23 – 25, pages 55–61, New Delhi, India.*

Shi, C., S.-Y. Wang, and B. Bhargava: 1999, 'MPEG Video Encryption in Real-time Using Secret Key Cryptography'. In: *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99), Las Vegas, Nevada, USA.*

Stinson, D. R.: 1995, *Cryptography Theory and Practice.* New York: CRC Press, Inc.

Tang, L.: 1996, 'Methods for Encrypting and Decrypting MPEG Video Data Efficiently'. *In Proceedings of the ACM Multimedia96, pages 219-229, Boston, MA.*

Zeng, W. and B. Liu: 1995, 'Geometric-structure-based Directional Filter for error Concealment in image/video transmission'. *Proc. SPIE Wireless Data Transmission at Information Systems/Photonics East'95, vol 2601, pp. 145-156.*

Appendix

The cosine transform of a 8×8 block is defined as:

$$F(u,v) = \frac{C(u)C(v)}{8} \sum_{x=0}^{7} \sum_{y=0}^{7} I(x,y) \cos \frac{(2x+1)u\pi}{16} \cos \frac{(2y+1)v\pi}{16}$$

where $u, v = 0, 1, \cdots, 7$, and

$$C(u), C(v) = \left\{ \begin{array}{ll} \frac{1}{\sqrt{2}}, & u,v = 0 \\ 1, & otherwise \end{array} \right\}$$

The sum of AC coefficients ($u = v =\neq 0$) of a block can be obtained as:

$$\begin{aligned} \sum_{u=0}^{7} \sum_{y=0}^{7} F(u,v) &= \sum_{u=0}^{7} \sum_{y=0}^{7} \left[\frac{C(u)C(v)}{8} \sum_{x=0}^{7} \sum_{y=0}^{7} I(x,y) \cos \frac{(2x+1)u\pi}{16} \cos \frac{(2y+1)v\pi}{16} \right] \\ &= \sum_{x=0}^{7} \sum_{y=0}^{7} \left[\sum_{u=1}^{7} \sum_{v=0}^{7} I(x,y) \cos \frac{(2x+1)u\pi}{16} \cos \frac{(2y+1)v\pi}{16} \right] \\ &= \sum_{x=0}^{7} \sum_{y=0}^{7} I(x,y) + \sum_{x=0}^{7} \sum_{y=0}^{7} I(x,y) \cdot \alpha \\ &= F(0,0) \cdot (1+\alpha) \end{aligned}$$

where α is the sum of a set of cosine function. One can find that the DC coefficient of a block can be derived from the sum of its AC coefficients.