



**INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO**



Cryptography

“Hill cipher with operation modes”

Abstract

In this document we explain the hill cipher, the different modes of operation and the implementation of the modes using hill to encrypt and decrypt bmp images

By:

Garduño Velazquez Alan

Professor:

MSc. NIDIA ASUNCIÓN CORTEZ DUARTE

October 2017

I appear in the photo of the
board

Index

Introduction:.....	1
Literature review:.....	1
Software (libraries, packages, tools):.....	4
Procedure:.....	4
Results.....	5
Discussion:.....	7
Conclusions:.....	8
References:	8
Code.....	9

Introduction:

Hill cipher is also a classical cipher, but this time is poly alphabetic cipher that use linear algebra to work.

In this work we are going to explain the hill cipher, the operations modes, their implementations using the hill cipher to encrypt, decrypt bmp images showing the transformation of the image with each operation mode, also we will describe the mathematic fundamentals of the hill cipher and explain how do the operations modes work.

Literature review:

Hill cipher is classical cipher that works with multiple alphabets and his mathematical fundamentals are based in linear algebra.

For encrypt a message using the hill cipher you will need the message to encrypt and a key, for this cipher the plain text will be represented in a matrix ($M \times M$) and the key in matrix ($M \times M$), in this case we will encrypt images (bmp) the pixels can take values between 0-255 and to encrypt we just must multiply the matrixes and apply module 256, but before encrypting we must be sure that the matrix K is an inversible matrix, that mean that its determinant must be different of 0, if the key matrix is not inversible will be impossible to decrypt the messages So the formula is the following:

$$C = (mxk) \bmod 256$$

Where C is the matrix of the ciphered message, m is the matrix of the plain mesasage and k is the matrix of the key, in linear algebra we must multiplicate files of the matrix m x columns of the matrix K to build a new matrix, in this case C .

To decrypt we need to obtain the inverse matrix of the key then we must multiplicate the inverse matrix of and C (the matrix resulted of the encryption) and after that we must apply module 26. So, the formula is the following.

$$m = (Cxk^{-1}) \bmod 256$$

Now we are to talk about the operation modes. The operation modes basically are algorithms that use a cipher as a part of the algorithm process, the cipher could be anyone in this case we use the hill cipher as a part of the process of the modes of operation, so we will put the diagrams and the formulas to encrypt and decrypt using the modes.

ECB

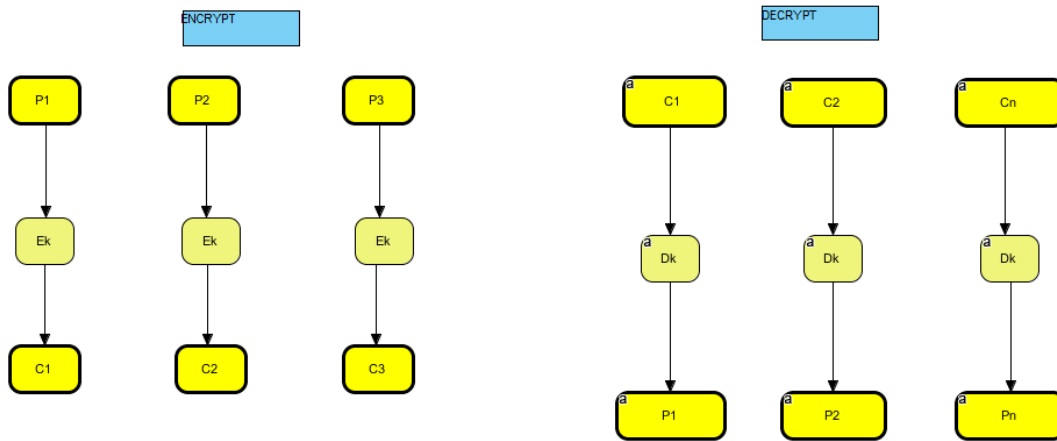


Figure 1, ECB encrypt and decrypt diagram.

Formula to encrypt using ECB: $C_n = E_k(P_n)$

Formula to decrypt using ECB: $P_n = D_k(C_n)$

CBC

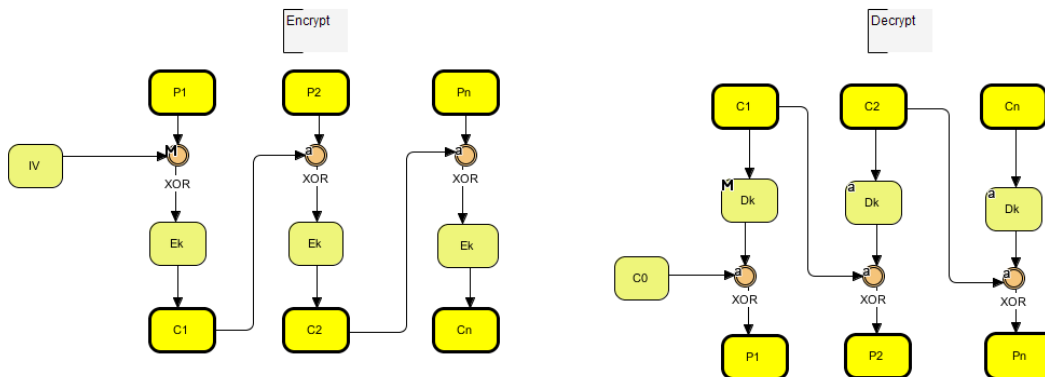


Figure 2, CBC encrypt and decrypt diagram

Formula to encrypt using CBC: $C_n = E_k(P_n \text{ xor } C_{n-1})$

Formula to decrypt using CBC: $P_n = D_k(C_n) \text{ xor } C_{n-1}$

CFB

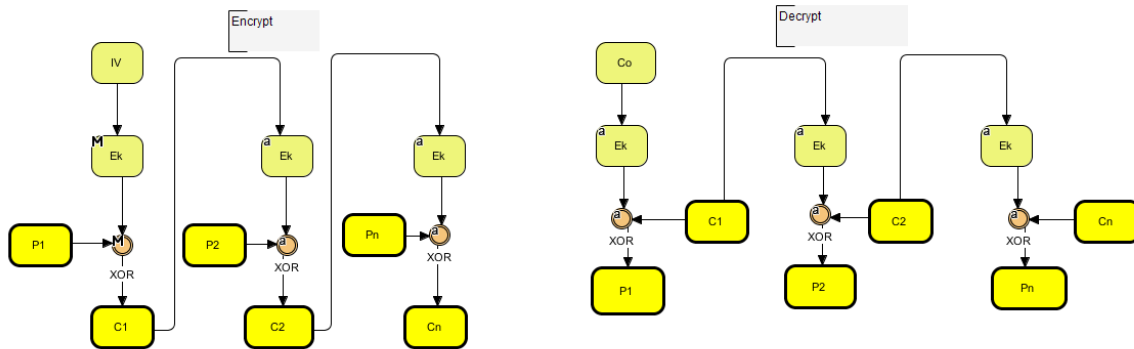


Figure 3, CFB encrypt and decrypt diagram

Formula to encrypt using CFB: $C_n = E_k(C_{n-1}) \text{ xor } P_n$

Formula to decrypt using CFB: $P_n = C_n \text{ xor } E_k(C_{n-1})$

OFB

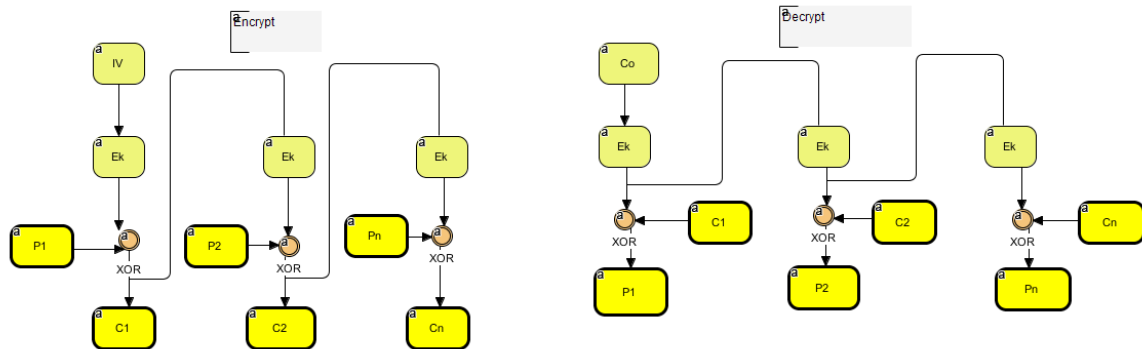


Figure 4, OFB encrypt and decrypt diagram

Formula to encrypt using OFB: $C_n = P_n \text{ xor } E_k(C_{n-1})$

Formula to decrypt using OFB: $P_n = C_n \text{ xor } E_k(P_{n-1})$

In this case all the modes use the hill cipher in the block of E_k and D_k . In the E_k block the modes use the formula $C=(mxk)\text{mod}256$ and in the block of D_k use the formula $m=(Cxxk^{(-1)})\text{mod}256$ explained before.

Software (libraries, packages, tools):

Libraries¹:

<stdlib.h>

For manege dinamic memory , the used function

- Malloc()

<stdio.h>

For standart ouput and input and manege files, the used function were:

- Scanf()
- Printf()
- Fprintf()
- Fgetc()
- Fopen()

Tools

- Sublime text 3
- Visual Paradigm 14.2

Procedure:

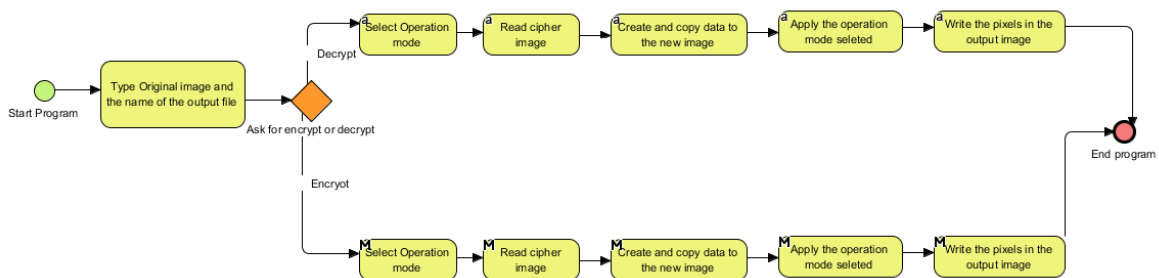


Figure 5, Flow chart

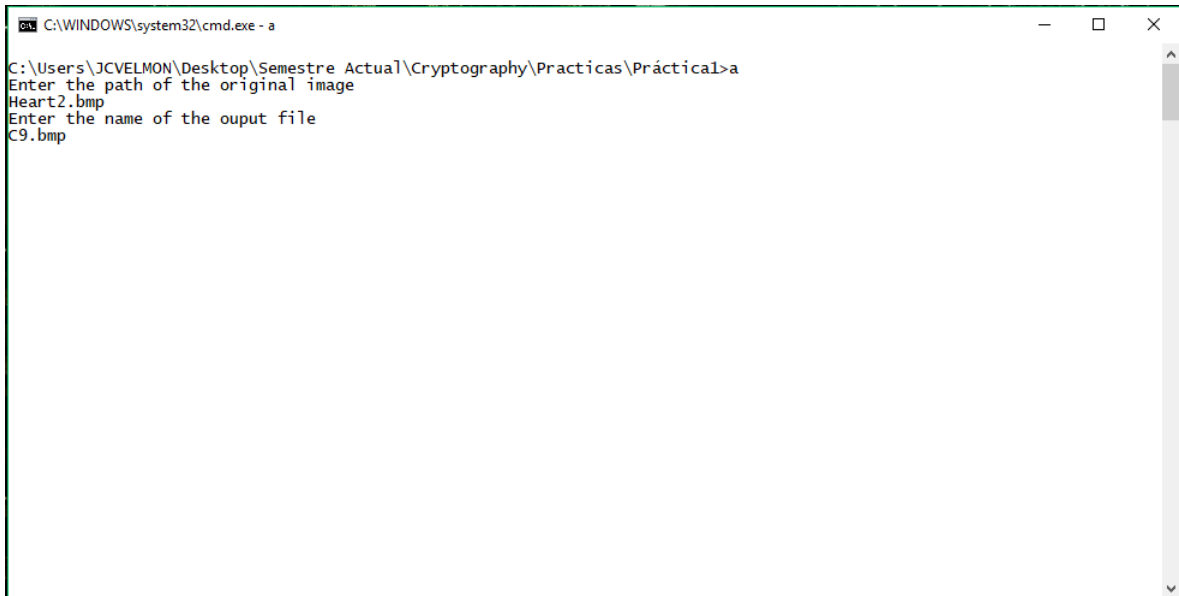
That we do is request a user the path of the original image and the path of the output image, then ask for encrypt or decrypt the image, then ask for which mode of operation does the user want to use.

Then when we have the path of the images we copy the information to the output image except the pixels, then read the pixels of the original image and apply the hill cipher according to the operation modes that was selected and write the cipher pixels in the output image.

Results

Now we are going to see the results of the implementation:

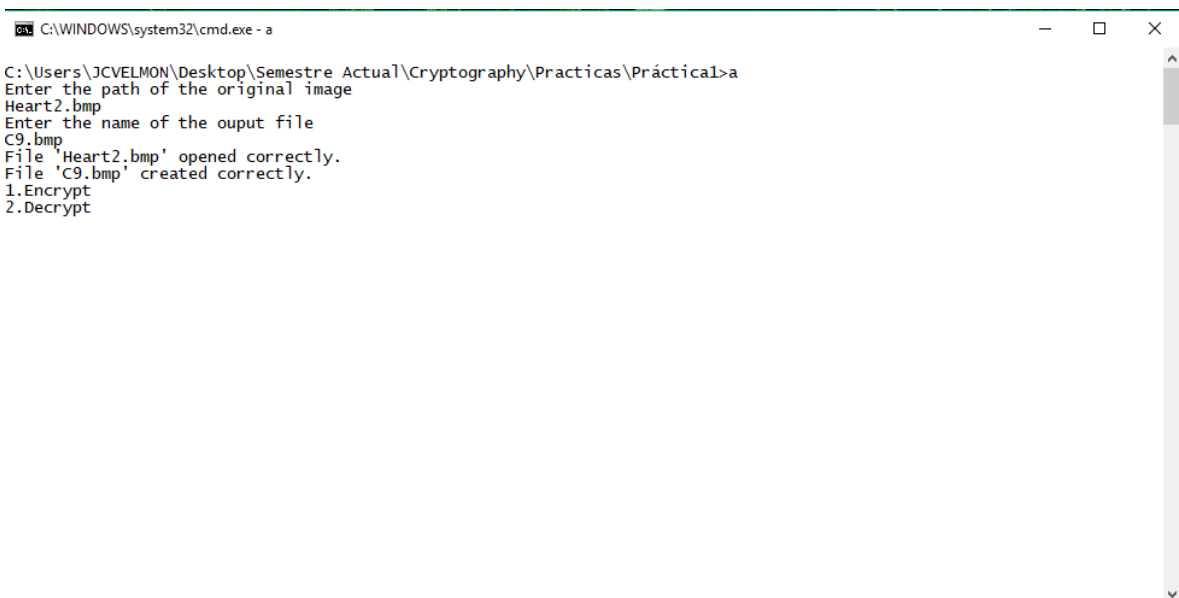
First, requesting the path of the original image and name of the output file.



```
C:\WINDOWS\system32\cmd.exe - a
C:\Users\JCVELMON\Desktop\Semestre Actual\Cryptography\Practicas\Práctical>a
Enter the path of the original image
Heart2.bmp
Enter the name of the ouput file
C9.bmp
```

Figure 6, requesting the path and the name of the output file.

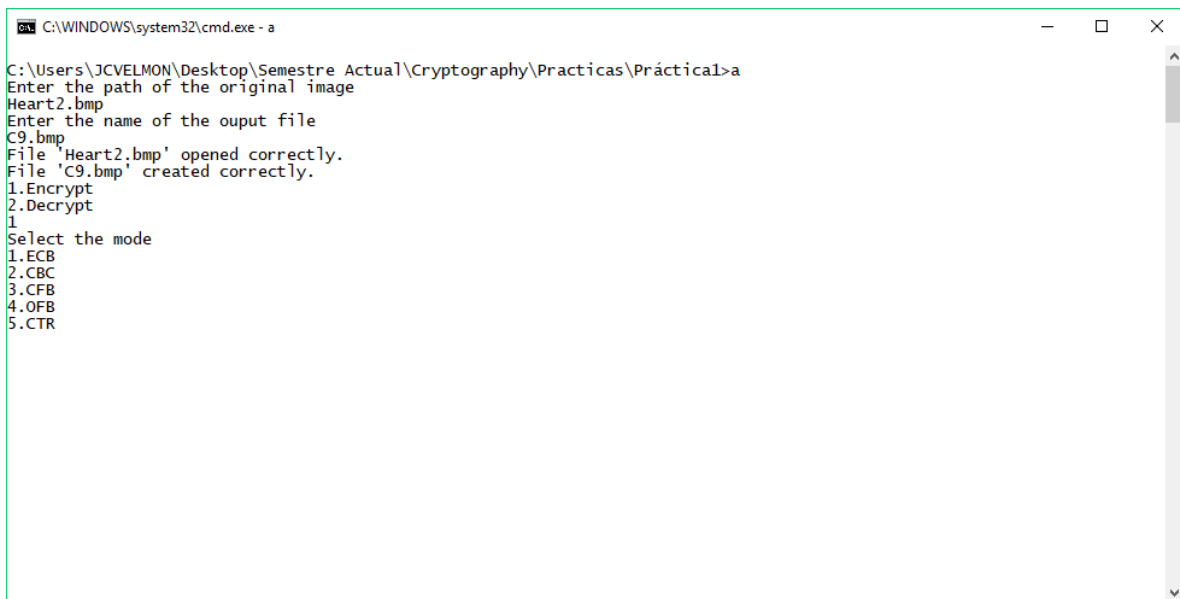
Then we ask for encrypt or decrypt: (View Figure 7)



```
C:\WINDOWS\system32\cmd.exe - a
C:\Users\JCVELMON\Desktop\Semestre Actual\Cryptography\Practicas\Práctical>a
Enter the path of the original image
Heart2.bmp
Enter the name of the ouput file
C9.bmp
File 'Heart2.bmp' opened correctly.
File 'C9.bmp' created correctly.
1.Encrypt
2.Decrypt
```

Figure 7, ask for incrypt or decrypt

Then we ask to the user which mode of operation does the user want to use: (View Figure 8)



```
C:\WINDOWS\system32\cmd.exe - a
C:\Users\JCVELMON\Desktop\Semestre Actual\Cryptography\Practicas\Practical>a
Enter the path of the original image
Heart2.bmp
Enter the name of the output file
C9.bmp
File 'Heart2.bmp' opened correctly.
File 'C9.bmp' created correctly.
1.Encrypt
2.Decrypt
3
1
Select the mode
1.ECB
2.CBC
3.CFB
4.OFB
5.CTR
```

Figure 8, select the operation mode.

In the next table I will show what happens to the same images using all the modes (View Table 1)

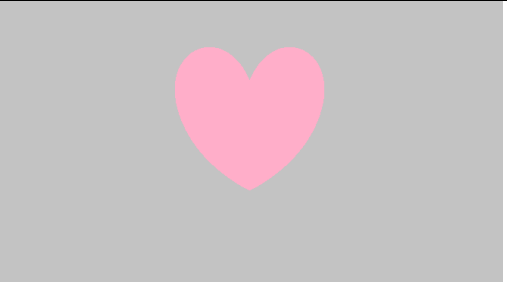
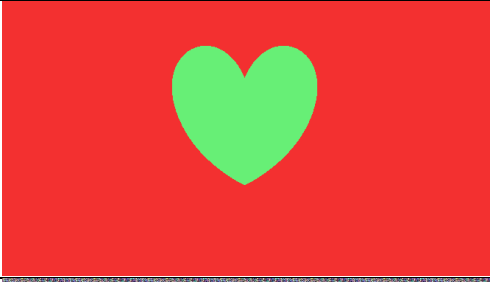
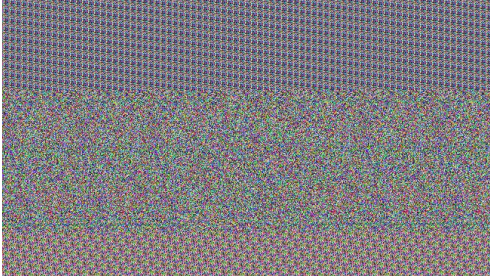
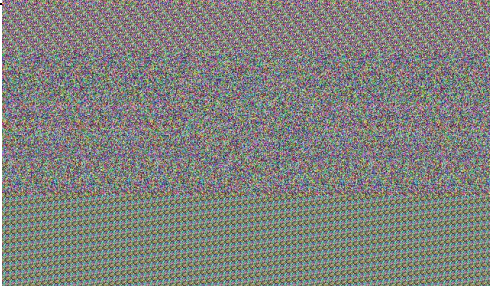
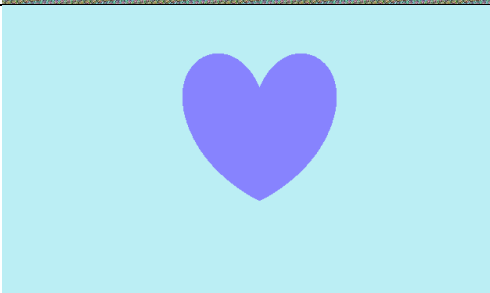
Mode	Original	Encrypted
ECB		
CBC		
CFB		
OFB		

Table 1

Discussion:

As we can see in the Table 1 with the cipher ECB and OFB the original image just changes the colors, and in the CFB and CBC are impossible to see something it important to mention that in all the images we use a predefined key, so all the images were ciphered with the same key, but the output are different because of the mode of operations that are different. Besides that, the output is different because as you can see in the literature review the modes follows some different steps and that steps are a little change in the process we that little changes have lot of influence in the output. Also this

modes improve a lot the security of the cipher, I mean the hill cipher is not pretty much secure so if we implement the hill cipher or another cipher, using CBC or CFB will be more secure.

Conclusions:

During this practice I learned how implement the operations modes and how it works on images I never have worked whit images, it was interesting learn how to read a bmp image in C language, this implementation could apply if you want to hide your private photos or a sensible photo of you or other persons.

This program can't read a different type of images, only can red 24-bit bmp images, so if the image is a bmp but not of 24 bits, the program will work but no correctly it will be impossible to decrypt the images.

This implementation has a lot of defects but the most significative to the results is if the image is a 24-bit bmp image, but the weight of the image not meet the condition of weight mode $256 = 4$, the output image will be unable to decrypt and will have noise in the output.

References:

- [1] Cplusplus.com. (2000). *Reference - C++ Reference*. [online] Available at: <http://www.cplusplus.com/reference/> [Accessed 9 Oct. 2017].
- [2] Poesía Binaria. (2017). *Leyendo archivos de imagen en formato BMP en C - Poesía Binaria*. [online] Available at: <https://totaki.com/poesiabinaria/2011/06/leyendo-archivos-de-imagen-en-formato-bmp-en-c/> [Accessed 9 Oct. 2017].
- [3] structure, r. (2017). *read bitmap file into structure*. [online] Stackoverflow.com. Available at: <https://stackoverflow.com/questions/14279242/read-bitmap-file-into-structure> [Accessed 9 Oct. 2017].
- [4] Edgardo Adrian Franco Martinez (2017). *Leer y escribe imagenes BMP* [online] Available at: <http://www.eafranco.com/docencia/sistemasoperativosii/files/programas/BMP.c> [Accessed 9 Oct. 2017].

Code

Note: The code To read and write in bmp structure was taken from the references 2 ,3 and 4

```
1. #include<stdio.h>
2. #include<stdlib.h>
3. #include "modes.c"
4.
5. int main( void ){
6.     FILE *org, *ciph;
7.     int op,mode;
8.     char*path = (char*)malloc(sizeof(char));
9.     char *cipher =(char*)malloc(sizeof(char));
10.    bmp image;
11.    printf("Enter the path of the original image\n");
12.    scanf("%s",path);
13.    printf("Enter the name of the ouput file\n");
14.    scanf("%s",cipher);
15.    org = open_file(path,cipher,1);
16.    ciph = open_file(path,cipher,2);
17.    read_head(org,ciph,&image);
18.    printf("1.Encrypt\n2.Decrypt\n");
19.    scanf("%d",&op);
20.    switch( op ){
21.        case 1:
22.            printf("Select the mode\n1.ECB \n2.CBC \n3.CFB \n4.OFB\n5.CTR\n");
23.            scanf("%d",&mode);
24.            if(mode == 1)
25.                ECB(org,ciph,&image,1);
26.            else if( mode == 2)
27.                CBC(org,ciph,&image,1);
28.            else if(mode == 3)
29.                CFB(org,ciph,&image,1);
30.            else if(mode == 4)
31.                OFB(org,ciph,&image,1);
32.            else if(mode == 5)
33.                CTR(org,ciph,&image,1,3);
34.            else
35.                {printf("No aviable option\n"); exit(0);}
36.        break;
37.        case 2:
38.            printf("Select the mode\n1.ECB \n2.CBC \n3.CFB \n4.OFT\n5.CTR\n");
39.            scanf("%d",&mode);
40.            if(mode == 1)
41.                ECB(org,ciph,&image,0);
42.            else if( mode == 2)
43.                CBC(org,ciph,&image,0);
44.            else if(mode == 3)
45.                CFB(org,ciph,&image,0);
46.            else if(mode == 4)
47.                OFB(org,ciph,&image,0);
48.            else if(mode == 5)
49.                CTR(org,ciph,&image,0,3);
50.            else
51.                {printf("No aviable option\n"); exit(0);}
52.        break;
```

```

53.         default:
54.             printf("No option available\n");
55.             exit(0);
56.         break;
57.     }
58.     return 0;
59. }

```

```

1. #include<stdlib.h>
2.
3. typedef struct BMP
4. {
5.     char type [2];
6.     int file_size;
7.     int reserved;
8.     int offset;
9.     int bitmap_size;
10.    int width;
11.    int height;
12.    short no_planes;
13.    short bits_per_pixel;
14.    int compression;
15.    int image_size;
16.    int horizontal_res;
17.    int vertical_res;
18.    int no_colors;
19.    int important_colors;
20. }bmp;
21.
22. typedef struct llave
23. {
24.     unsigned char Ek [3][3];
25.     unsigned char Dk [3][3];
26. }llave;
27.
28. llave key =
29. {
30.     {
31.         {1, 2, 3},
32.         {4, 5, 6}, //key
33.         {11, 9, 8}
34.     },
35.     {
36.         {90, 167, 1}, //Iverse of the key mod 256
37.         {74, 179, 254},
38.         {177, 81, 1}
39.     }
40. };
41.
42. FILE * open(char * original, char * encrypted, int tipo);
43. void copy_information(FILE * original, FILE * encrypted, bmp * image);
44. unsigned char* hill(unsigned char * BMP, int mode);
45. void ECB(FILE* org, FILE* ciph, bmp* image, int mode);

```

```

46. void CBC(FILE* org, FILE* ciph, bmp* image, int mode);
47. void CFB(FILE* org, FILE* ciph, bmp* image, int mode);
48. void OFB(FILE* org, FILE* ciph, bmp* image, int mode);
49. void CTR(FILE* org, FILE* ciph, bmp* image, int counter);

```

```

1. unsigned char* hill(unsigned char * BMP, int mode){
2.     int i;
3.     unsigned char* res[3] = (unsigned char*)malloc(sizeof(char*3))
4.
5.     for (i = 0; i < 3; i ++){
6.         {
7.             if (mode == 0)
8.                 res [i] = ((BGR [0] * key.Dk [0][i]) + (BGR [1] * key.Dk [1][i]) + (BGR [2] * key.Dk [2][i])) % 256;
9.             else
10.                res [i] = ((BGR [0] * key.Ek [0][i]) + (BGR [1] * key.Ek [1][i]) + (BGR [2] * key.Ek [2][i])) % 256;
11.        }
12.        return res;
13.    }
14.
15. void ECB(FILE * org, FILE * ciph, bmp * image, int mode){
16.     for(i = 0; i < (image->image_size); i++){
17.         fread(&BGR, sizeof(char),3,org);
18.         fwrite(hill((unsigned char*)BGR,mode), sizeof(char), 3, ciph);
19.     }
20.
21. }
22. void CBC(FILE * org, FILE * ciph, bmp * image, int mode){
23.     unsigned char *res,pixel[3],BGR[3];
24.     if (mode == 1)
25.     {
26.         printf ("\n Type Co:\n");
27.         scanf ("%u %u %u", &pixel [0], &pixel [1], &pixel [2]);
28.         for (i = 0; i < (image -> image_size); i ++){
29.             {
30.                 fread (&BGR, sizeof (char), 3, org);
31.                 for (j = 0; j < 3; j ++){
32.                     BGR [j] = (pixel [j] ^ BGR [j]);
33.                     res = hill ((unsigned char * ) BGR, mode);
34.                     fwrite (res, sizeof (char), 3, ciph);
35.                 }
36.             }else
37.             {
38.                 printf ("\n Type Co:\n");
39.                 scanf ("%u %u %u", &pixel [0], &pixel [1], &pixel [2]);
40.                 for (i = 0; i < (image -> image_size); i ++){
41.                     {
42.                         fread (&BGR, sizeof (char), 3, org);
43.                         hill ((unsigned char * ) BGR, (unsigned char * ) aux, mode);
44.                         for (j = 0; j < 3; j ++){
45.                             pixel [j] = (pixel [j] ^ aux [j]);
46.                             fwrite (&pixel, sizeof (char), 3, ciph);

```

```

47.         for (j = 0; j < 3; j++)
48.             pixel[j] = BGR[j];
49.     }
50. }
51. }
52. void CFB(FILE * org, FILE * ciph, bmp * image, int mode){
53.     unsigned char *res,pixel[3],BGR[3];
54.     unsigned char cfb[3];
55.     if (mode == 1)
56.     {
57.         printf ("\n Type Co:\n");
58.         scanf ("%u %u %u", &pixel [0], &pixel [1], &pixel [2]);
59.         for (i = 0; i < (image -> image_size); i++)
60.         {
61.             fread (&BGR, sizeof (char), 3, org);
62.             hill ((unsigned char * )cfb, (unsigned char * )aux, mode);
63.             for (j = 0; j < 3; j++)
64.                 cfb[j] = (aux[j] ^ BGR[j]);
65.             fwrite (&cfb, sizeof (char), 3, ciph);
66.         }
67.     }else
68.     {
69.         printf ("\n Type Co:\n");
70.         scanf ("%u %u %u", &pixel [0], &pixel [1], &pixel [2]);
71.         for (i = 0; i < (image -> image_size); i++)
72.         {
73.             hill ((unsigned char * )pixel, (unsigned char * ) aux, 1);
74.             fread (&BGR, sizeof (char), 3, org);
75.             for (j = 0; j < 3; j++)
76.                 cfb[j] = (aux[j] ^ BGR[j]);
77.             fwrite (&cfb, sizeof (char), 3, ciph);
78.         }
79.     }
80. }
81. void OFB(FILE * org, FILE * ciph, bmp * image, int mode){
82.     unsigned char *res,pixel[3],BGR[3];
83.     unsigned char aux2 [3];
84.     printf ("\n Type Co:\n");
85.     scanf ("%u %u %u", &pixel [0], &pixel [1], &pixel [2]);
86.     hill ((unsigned char * ) pixel, (unsigned char * ) aux, 1);
87.     for (i = 0; i < 3; i++)
88.         aux2[i] = aux[i];
89.     for (i = 0; i < (image -> image_size); i++)
90.     {
91.         fread (&BGR, sizeof (char), 3, org);
92.         for (j = 0; j < 3; j++)
93.             pixel[j] = (aux2[j] ^ BGR[j]);
94.         fwrite (res, sizeof (char), 3, ciph);
95.         res=hill ((unsigned char * ) aux, 1);
96.     }
97. }
98. }
99. void CTR(FILE * org, FILE * ciph, bmp * image, int counter, int mode){
100.     unsigned char *res,pixel[3],BGR[3];
101.     unsigned char aux2 [3];
102.     int k;
103.     printf ("\n Type Co:\n");
104.     scanf ("%u %u %u", &pixel [0], &pixel [1], &pixel [2]);
105.     hill ((unsigned char * ) pixel, (unsigned char * ) aux, 1);
106.     for( k = 0; k < counter; k++){
107.         for (i = 0; i < 3; i++)

```

```

108.         aux2 [i] = aux [i];
109.     for (i = 0; i < (image -> image_size); i++)
110.     {
111.         fread (&BGR, sizeof (char), 3, org);
112.         for (j = 0; j < 3; j++)
113.             pixel [j] = (aux2 [j] ^ BGR [j]);
114.         //We realize XOR between pixel and BGR from Image
115.         fwrite (res, sizeof (char), 3, ciph);
116.         res=hill ((unsigned char * ) aux, 1);
117.     }
118. }
119. }
120.
121. FILE * open(char * original, char * encrypted, int tipo)
122. {
123.     FILE * pt1, * pt2;
124.     //We open the file in binary mode to read
125.     pt1 = fopen (original, "rb");
126.     if (pt1 == NULL)
127.     {
128.         printf("Error while opening file: '%s'.\n", original);
129.         exit(0);
130.     }
131.     pt2 = fopen (encrypted, "wb");
132.     if (pt2 == NULL)
133.     {
134.         printf("Error while creating file: '%s'.\n", encrypted);
135.         exit(1);
136.     }
137.     if (tipo == 1)
138.     {
139.         printf("File '%s' opened correctly.\n", original);
140.         return pt1;
141.     }
142.     else
143.     {
144.         printf("File '%s' created correctly.\n", encrypted);
145.         return pt2;
146.     }
147. }
148.
149. void copy_information (FILE * original, FILE * encrypted, bmp * image)
150. {
151.     //Type (must be 'BM')
152.     fread (&image -> type, sizeof (char), 2, original);
153.     fwrite (&image -> type, sizeof (char), 2, encrypted);
154.
155.     //Size of the file
156.     fread (&image -> file_size, sizeof (int), 1, original);
157.     fwrite (&image -> file_size, sizeof (int), 1, encrypted);
158.
159.     //Reserved bytes
160.     fread (&image -> reserved, sizeof (int), 1, original);
161.     fwrite (&image -> reserved, sizeof (int), 1, encrypted);
162.
163.     //Offset
164.     fread (&image -> offset, sizeof (int), 1, original);
165.     fwrite (&image -> offset, sizeof (int), 1, encrypted);
166.
167.     //Size of the bitmap

```

```

168.         fread (&image -> bitmap_size, sizeof (int), 1, original);
169.         fwrite (&image -> bitmap_size, sizeof (int), 1, encrypted);
170.
171.         //Width
172.         fread (&image -> width, sizeof (int), 1, original);
173.         fwrite (&image -> width, sizeof (int), 1, encrypted);
174.
175.         //Height
176.         fread (&image -> height, sizeof (int), 1, original);
177.         fwrite (&image -> height, sizeof (int), 1, encrypted);
178.
179.         //Number of planes
180.         fread (&image -> no_planes, sizeof (short),1, original);
181.         fwrite (&image -> no_planes, sizeof (short),1, encrypted);
182.
183.         //Bits per pixel
184.         fread (&image -> bits_per_pixel, sizeof (short),1, original);
185.         fwrite (&image -> bits_per_pixel, sizeof (short),1, encrypted);
186.
187.         //Type of compression (must be 0)
188.         fread (&image -> compression, sizeof (int), 1, original);
189.         fwrite (&image -> compression, sizeof (int), 1, encrypted);
190.
191.         //Size of the image
192.         fread (&image -> image_size, sizeof (int), 1, original);
193.         fwrite (&image -> image_size, sizeof (int), 1, encrypted);
194.
195.         //Horizontal resolution
196.         fread (&image -> horizontal_res, sizeof (int), 1, original);
197.         fwrite (&image -> horizontal_res, sizeof (int), 1, encrypted);
198.
199.         //Vertical resolution
200.         fread (&image -> vertical_res, sizeof (int), 1, original);
201.         fwrite (&image -> vertical_res, sizeof (int), 1, encrypted);
202.
203.         //Number of colors
204.         fread (&image -> no_colors, sizeof (int), 1, original);
205.         fwrite (&image -> no_colors, sizeof (int), 1, encrypted);
206.
207.         //Number of important colors
208.         fread (&image -> important_colors, sizeof (int), 1, original);
209.         fwrite (&image -> important_colors, sizeof (int), 1, encrypted);
210.
211.         //We check if the selected file is a bitmap
212.         if (image -> type [0] != 'B' || image -> type [1] != 'M')
213.         {
214.             printf ("The image must be a bitmap.\n");
215.             exit (1);
216.         }
217.         if (image -> bits_per_pixel != 24)
218.         {
219.             printf ("The image must be 24-bits.\n");
220.             exit (1);
221.         }
222.     }

```