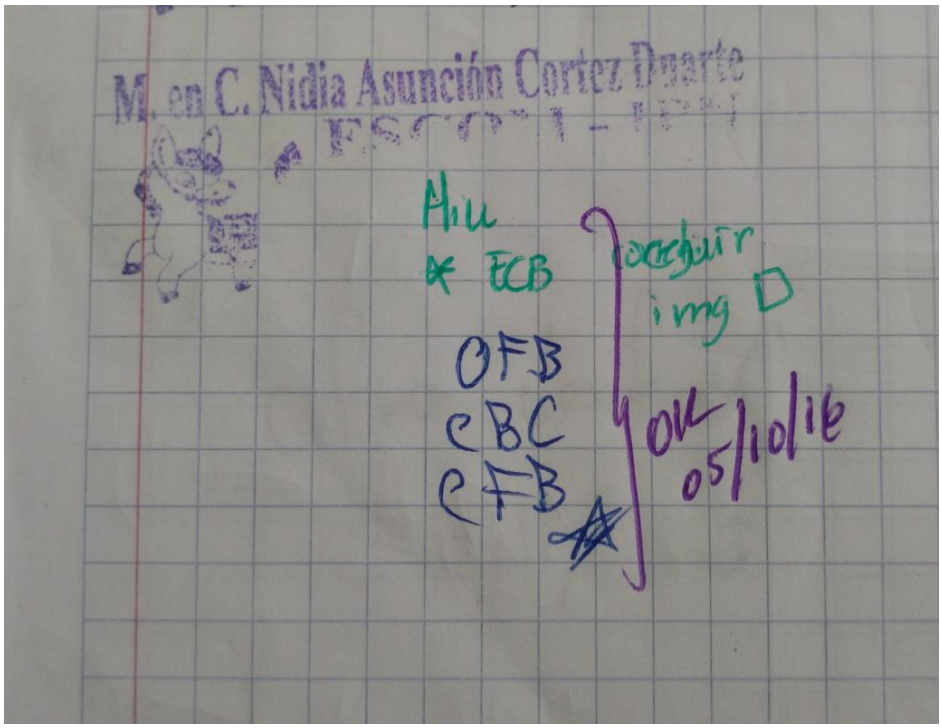
Instituto Politécnico Nacional
Escuela Superior de Cómputo
Cryptography

Practice 4: Hill Cipher and operation modes.

By:
Moreno Zárate Víctor Gibrán

Professor:
M. en C. NIDIA ASUNCIÓN CORTEZ DUARTE
September 2016



Contenido

Problem

What is the behavior of the Hill cipher?

What are the main characteristics of the Hill cipher?

How does the output behave per every mode of operation?

A block cipher is a function which maps n-bit plaintext blocks to n-bit ciphertext blocks; n is called the block length. It may be viewed as a simple substitution cipher with large character size. The function is parameterized by a k-bit key *K*, 1 taking values from a subset K (the key space) of the set of all k-bit vectors V_k. It is generally assumed that the key is chosen at random. Use of plaintext and ciphertext blocks of equal size avoids data expansion. [1]

The Hill Cipher is an example of polyalphabetical cipher that employs the modulus operation and techniques of linear algebra. They key k used in this cipher is an *nxn* matrix of integers. Encryption is done by transforming a group P of n symbols over the plain text into another group C of n symbols over the cipher alphabet. [2]

The encryption is $C = K * P \ \ mod\ m$ and decryption $P = K^{-1} * \ C\ mod\ n$

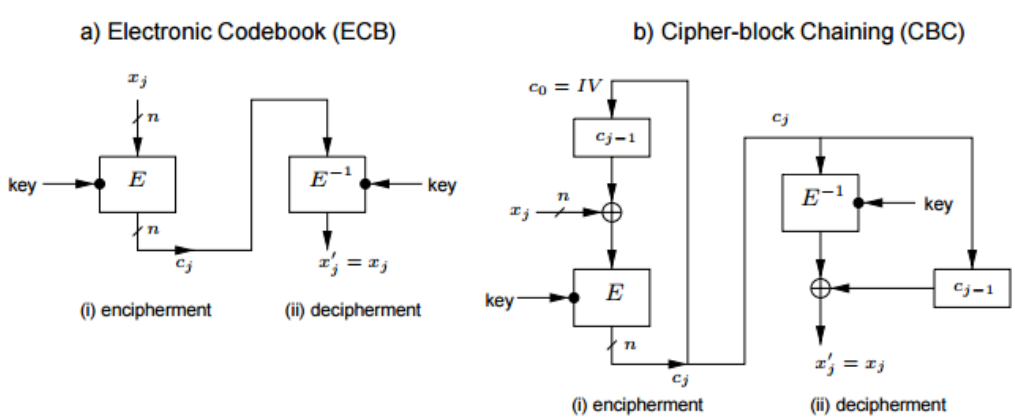
A block cipher encrypts plaintext in fixed-size n-bit blocks (often n = 64). For messages, exceeding n bits, the simplest approach is to partition the message into n-bit blocks and encrypt each separately. This electronic-codebook (ECB) mode has disadvantages in most applications, motivating other methods of employing block ciphers (modes of operation) on larger messages. The four most common modes are ECB, CBC, CFB, and OFB. [1]

CBC: $Cj = E\left(P \oplus C_{j-1}\right)$
$Pj = D(Cj) \oplus C_{j-1}$

CFB: $Cj = E\left(C_{j-1}\right) \oplus Pj$
$Pj = E(C_{j-1}) \oplus Cj$

OFB: $Cj = Pj \oplus \ Oj$

$Pj = Cj \oplus \ Oj \ \ ; \ Oj = E(Ij)$



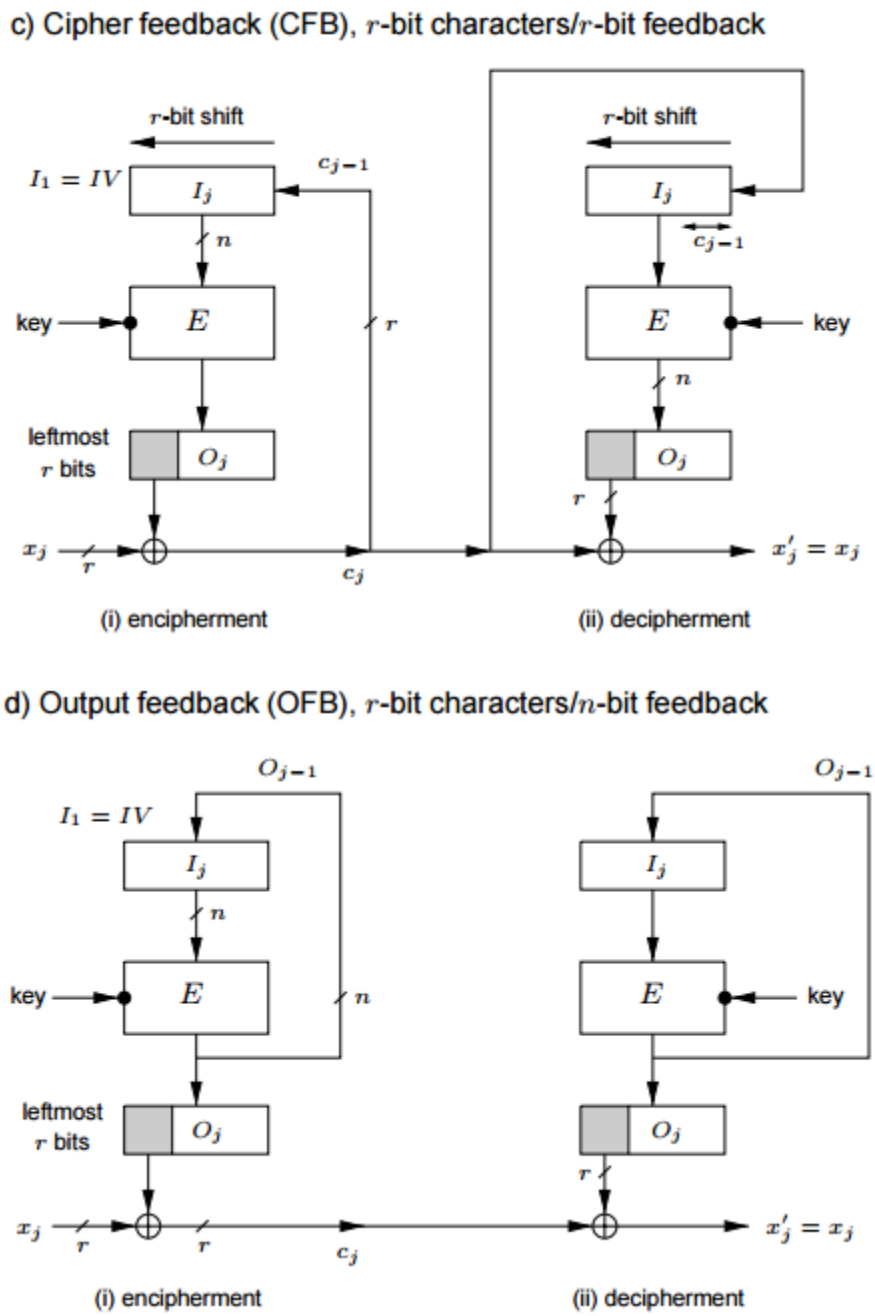


Illustration 1 Common modes of operation for an n-bit block cipher. [1]

Hypothesis

A possible solution to the problem is to write a computer program to analyze the behavior of the Hill cipher, this program will be able to encrypt and decrypt an image using this technique with all the modes of operation. The key, and the inverse key, will be hardcoded for this practice.

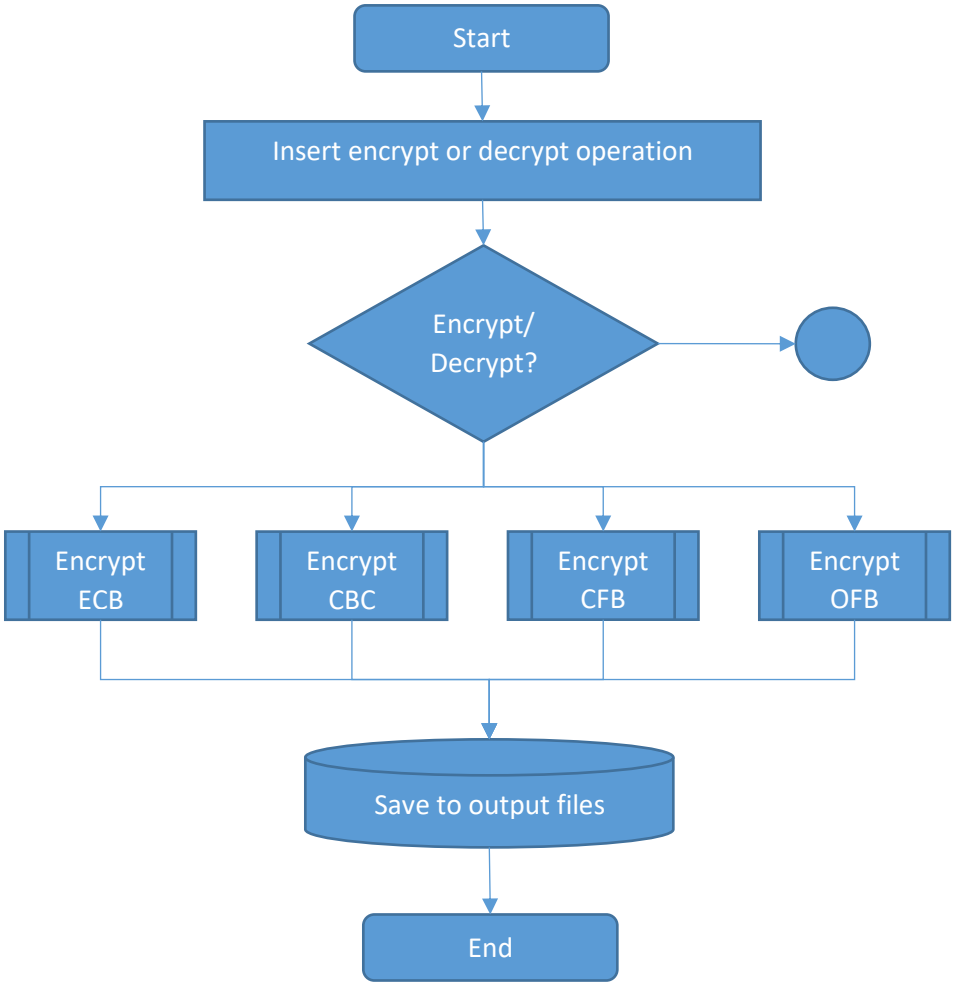
If we program this software correctly, we will see how the image's data changes per the mode of operation. Then we will observe the encrypt/decrypt of every resultant image as we put it in the correct mode.

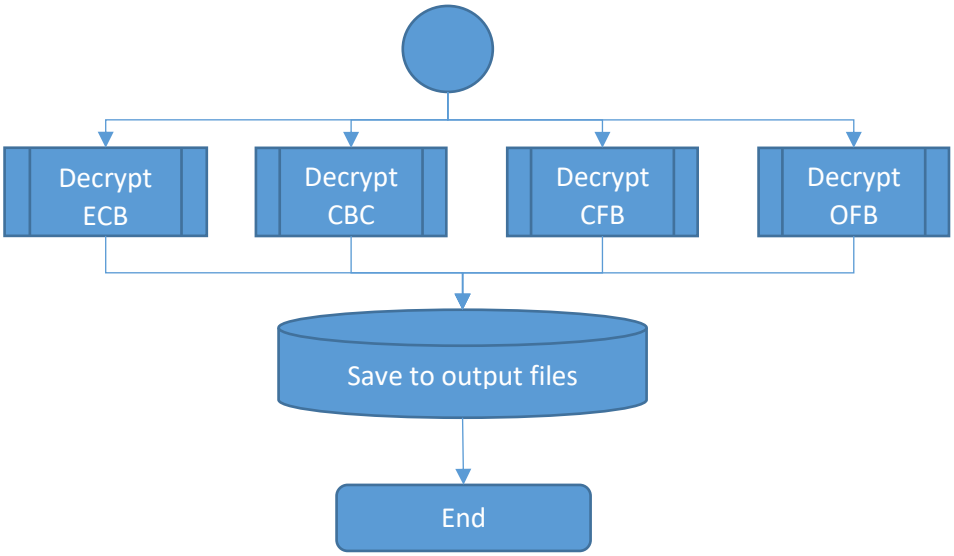
Software (libraries, package, tools)

In order to do this practice, we used:

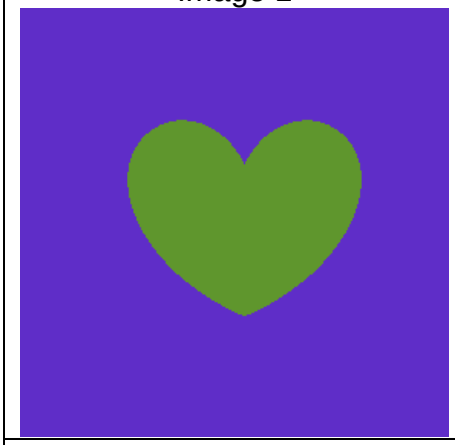
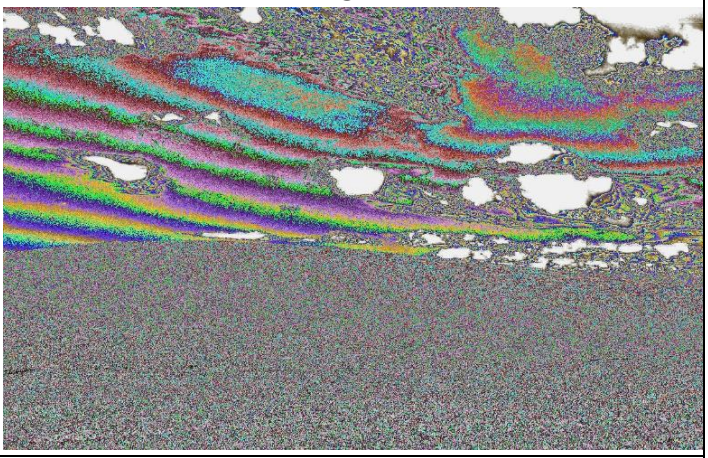
- Personal Computer
- Linux Operating System
- Image Editor
- Image Viewer
- GNU C Compiler

Procedure





Results (Data)

Image 1	Image 2
	
ECB	ECB
	

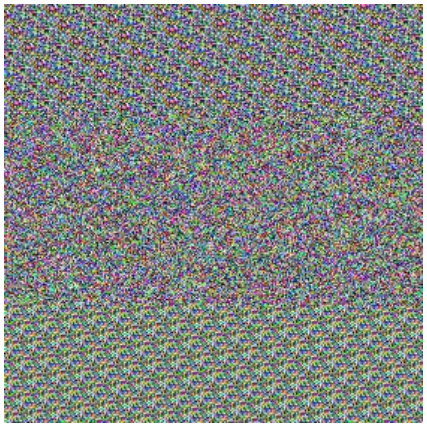
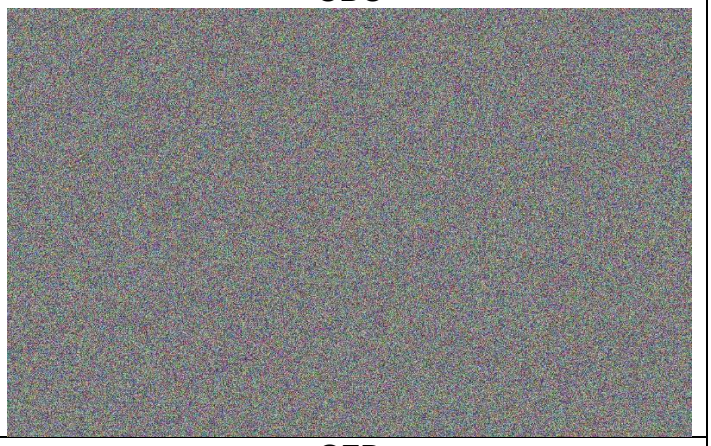
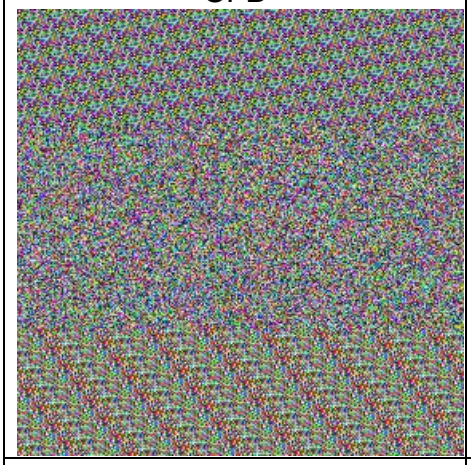
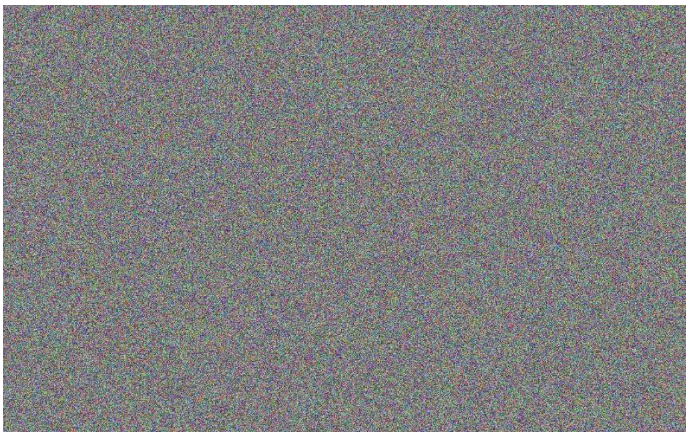
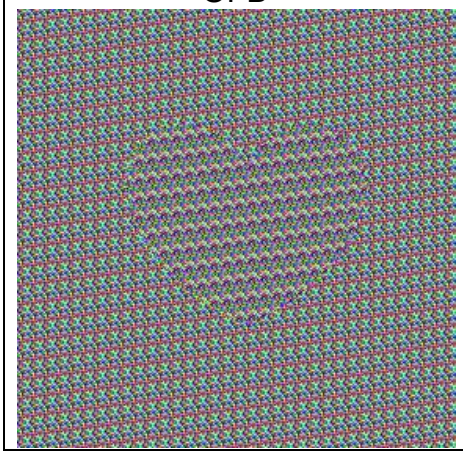
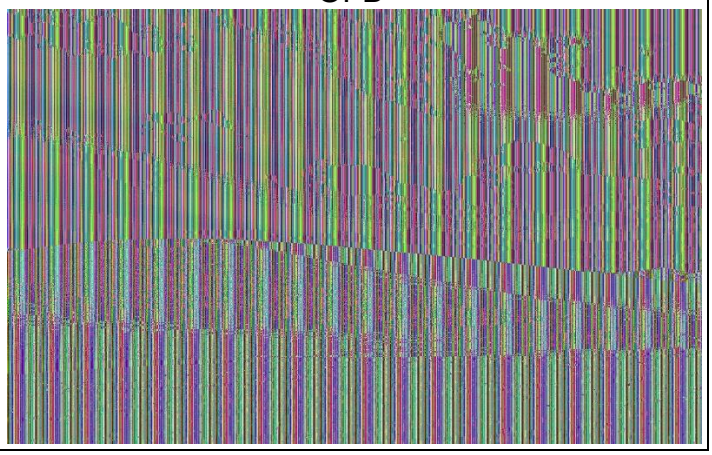
CBC 	CBC 
CFB 	CFB 
OFB 	OFB 

Table 1. Hill Cipher with different modes of operation

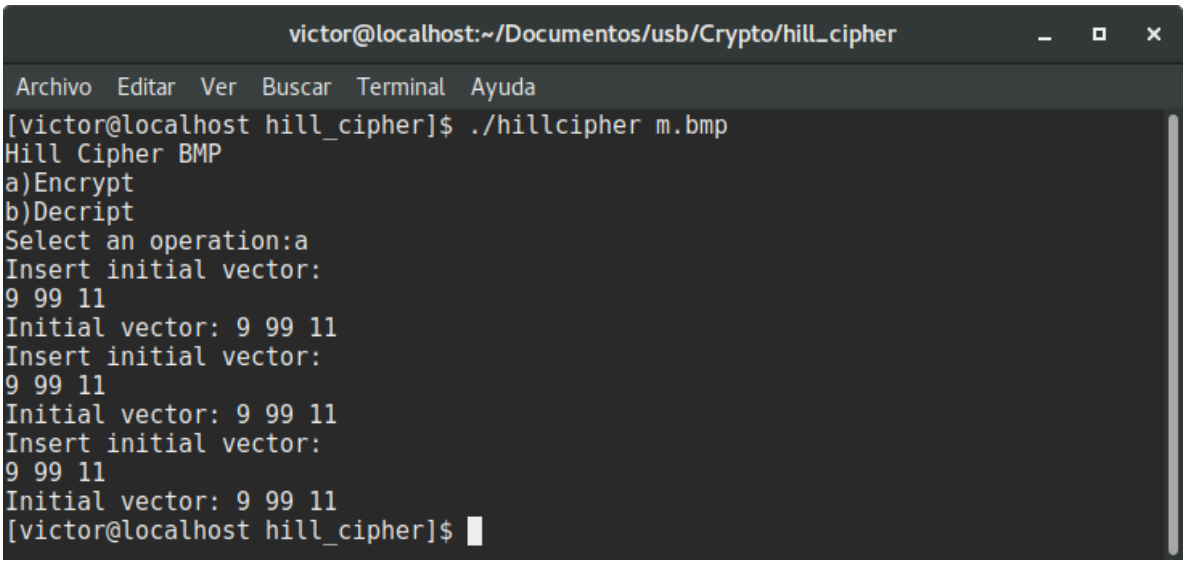


Figure 2. Encrypt process

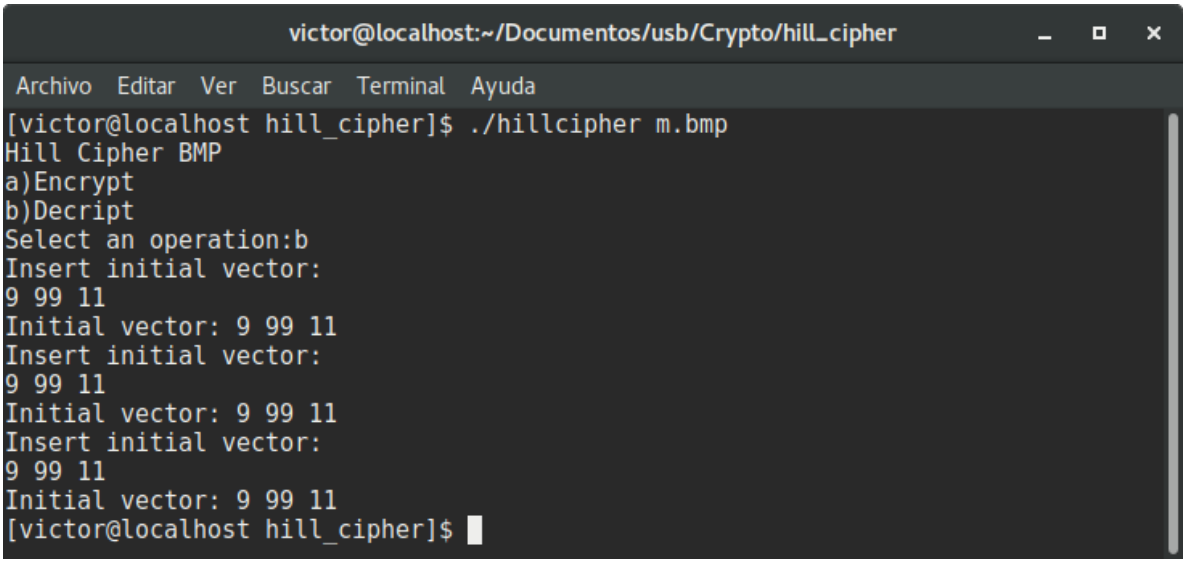


Figure 3. Decrypt process

Conclusions

In these practice, we code the Hill Cipher with different modes of operation, I observe that in the mode of operation ECB with the first image it only changed the present colors, and in the OFB mode, we can distinguish the shape of the original image. In this case, this mode with hill cipher are not secure at all. I also had a problem when BMP images that haven't shape of square, because of the offset.

I learn the use and implementation of this cryptographic method, where I used matrix multiplications and xors performed by a computer program, and it was an introduction to more complex methods. It can be used in real life to encrypt/decrypt simple BMP images.

Reference

[1] A. J. Menezes, P. C. Van Oorschot y S. A. Vanstone, Handbook of Applied Cryptography, CRC Press, 2001.

[2] D. Salomon, Data Privacy and Security, Springer Science & Business Media, 2012.

Code

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include "matrix.h"

void io(char filename[255],char output_filename[255],char opt);
void encrypt(FILE *file, FILE *file_dest);
void decrypt(FILE *file, FILE *file_dest);
void head_handler(FILE *file,FILE *file_dest);
void skip(FILE *file,FILE *file_dest,int n);
void hillCipher(int **matix_m,int **key);

void xor(int **matrix_m,int **block);

void encrypt_CBC(FILE *file, FILE *file_dest);
void encrypt_CFB(FILE *file, FILE *file_dest);
void encrypt_OFB(FILE *file, FILE *file_dest);

void decrypt_CFB(FILE *file, FILE *file_dest);
void decrypt_CBC(FILE *file, FILE *file_dest);
void decrypt_OFB(FILE *file, FILE *file_dest);

char input[255];

int main(int argc, char *argv[]){
      char opt;
      printf("Hill Cipher BMP\n");
      printf("a)Encrypt\nb)Decript\n");
      printf("Select an operation:");
      scanf("%c",&opt);
      strcpy(input,argv[1]);

      char filename[256],output_filename[256];

      io(filename,output_filename,opt);
      return 0;

      }
```

```
void io(char filename[256],char output_filename[256],char opt){
      FILE *file,*file_dest;


      if(opt=='a'){

            ///////////////////////ECB/////////////////////////////
            sprintf(filename,"%s",input);
            sprintf(output_filename,"%s","c_ecb.bmp");
            file = fopen(filename, "r");
            file_dest= fopen(output_filename,"w");

            if (file==NULL){
                  perror("Can't open file");
                  exit(0);
                  }

            if (file_dest==NULL){
                  perror("Can't create file");
                  exit(0);
                  }

            encrypt(file,file_dest);
            //Close Streams
            fclose(file);
            fclose(file_dest);

            ////////////////CBC/////////////////////////////////
            sprintf(filename,"%s",input);
            sprintf(output_filename,"%s","c_cbc.bmp");
            file = fopen(filename, "r");
            file_dest= fopen(output_filename,"w");

            if (file==NULL){
                  perror("Can't open file");
                  exit(0);
                  }

            if (file_dest==NULL){
                  perror("Can't create file");
                  exit(0);
                  }

            encrypt_CBC(file,file_dest);
            //Close Streams
            fclose(file);
            fclose(file_dest);

            ////////////CFB/////////////////
            sprintf(filename,"%s",input);
            sprintf(output_filename,"%s","c_cfb.bmp");
            file = fopen(filename, "r");
            file_dest= fopen(output_filename,"w");

            if (file==NULL){
                  perror("Can't open file");
                  exit(0);
```

```
            }

        if (file_dest==NULL){
            perror("Can't create file");
            exit(0);
            }

        encrypt_CFB(file,file_dest);
        //Close Streams
        fclose(file);
        fclose(file_dest);

        ////////////OFB/////////////////
        sprintf(filename,"%s",input);
        sprintf(output_filename,"%s","c_ofb.bmp");
        file = fopen(filename, "r");
        file_dest= fopen(output_filename,"w");

        if (file==NULL){
            perror("Can't open file");
            exit(0);
            }

        if (file_dest==NULL){
            perror("Can't create file");
            exit(0);
            }

        encrypt_OFB(file,file_dest);
        //Close Streams
        fclose(file);
        fclose(file_dest);





        }
    else if (opt=='b'){
        ///////////////////////ECB/////////////
        sprintf(filename,"%s","c_ecb.bmp");
        sprintf(output_filename,"%s","m_ecb.bmp");
        file = fopen(filename, "r");
        file_dest= fopen(output_filename,"w");

        if (file==NULL){
            perror("Can't open file");
            exit(0);
            }

        if (file_dest==NULL){
            perror("Can't create file");
            exit(0);
            }

        decrypt(file,file_dest);
        //Close Streams
```

```
fclose(file);
fclose(file_dest);

//////////////////CBC//////////////////////////////////
sprintf(filename,"%s","c_cbc.bmp");
sprintf(output_filename,"%s","m_cbc.bmp");
file = fopen(filename, "r");
file_dest= fopen(output_filename,"w");

if (file==NULL){
      perror("Can't open file");
      exit(0);
      }

if (file_dest==NULL){
      perror("Can't create file");
      exit(0);
      }

decrypt_CBC(file,file_dest);
//Close Streams
fclose(file);
fclose(file_dest);

//////////////////CFB//////////////////////////////////
sprintf(filename,"%s","c_cfb.bmp");
sprintf(output_filename,"%s","m_cfb.bmp");
file = fopen(filename, "r");
file_dest= fopen(output_filename,"w");

if (file==NULL){
      perror("Can't open file");
      exit(0);
      }

if (file_dest==NULL){
      perror("Can't create file");
      exit(0);
      }

decrypt_CFB(file,file_dest);
//Close Streams
fclose(file);
fclose(file_dest);

//////////////////OFB//////////////////////////////////
sprintf(filename,"%s","c_ofb.bmp");
sprintf(output_filename,"%s","o_cfb.bmp");
file = fopen(filename, "r");
file_dest= fopen(output_filename,"w");

if (file==NULL){
      perror("Can't open file");
      exit(0);
      }

if (file_dest==NULL){
```

```
                perror("Can't create file");
                exit(0);
                }

            decrypt_OFB(file,file_dest);
            //Close Streams
            fclose(file);
            fclose(file_dest);



            }
        else
            printf("Invalid option");

}

void encrypt(FILE *file, FILE *file_dest){
    head_handler(file,file_dest);

    int caracter=0;
    int **matrix_m;
    int **key;
    creador_matrices_dinamicas(&matrix_m,1,3);
    creador_matrices_dinamicas(&key,3,3);

    //Matrix
    key[0][0]=1;
    key[0][1]=2;
    key[0][2]=3;
    key[1][0]=4;
    key[1][1]=5;
    key[1][2]=6;
    key[2][0]=11;
    key[2][1]=9;
    key[2][2]=8;

    while((caracter=fgetc(file))!=EOF){
                matrix_m[0][0]=caracter;
                matrix_m[0][1]=fgetc(file);
                matrix_m[0][2]=fgetc(file);

                hillCipher(matrix_m,key);

                fputc(matrix_m[0][0],file_dest);
                fputc(matrix_m[0][1],file_dest);
                fputc(matrix_m[0][2],file_dest);
            }

    }

void hillCipher(int **matrix_m,int**key){

    int **result;
    creador_matrices_dinamicas(&result,1,3);

    multiplicacion_matrices(matrix_m,key,result,1,3,3);
```

```
	matrix_m[0][0]=result[0][0];
	matrix_m[0][1]=result[0][1];
	matrix_m[0][2]=result[0][2];

	}

void decrypt(FILE *file, FILE *file_dest){
	head_handler(file,file_dest);
	int caracter=0;
	int **matrix_m;
	int **inv_key;
	creador_matrices_dinamicas(&matrix_m,1,3);
	creador_matrices_dinamicas(&inv_key,3,3);

	//Matrix
	inv_key[0][0]=90;
	inv_key[0][1]=167;
	inv_key[0][2]=1;
	inv_key[1][0]=74;
	inv_key[1][1]=179;
	inv_key[1][2]=254;
	inv_key[2][0]=177;
	inv_key[2][1]=81;
	inv_key[2][2]=1;

	while((caracter=fgetc(file))!=EOF){
			matrix_m[0][0]=(unsigned char)caracter;
			matrix_m[0][1]=(unsigned char)fgetc(file);
			matrix_m[0][2]=(unsigned char)fgetc(file);

			hillCipher(matrix_m,inv_key);

			fputc(matrix_m[0][0],file_dest);
			fputc(matrix_m[0][1],file_dest);
			fputc(matrix_m[0][2],file_dest);
		}
	}



void head_handler(FILE *file,FILE *file_dest){
	int offset;
	skip(file,file_dest,10);
	fread(&offset,sizeof(int),1,file );
	fwrite(&offset,sizeof(int),1,file_dest);
	skip(file,file_dest,offset-14);
	}

void skip(FILE *file,FILE *file_dest,int n){
	unsigned char caracter;
	int i;
	for(i=0;i<n;i++){
			caracter=getc(file);
			fputc(caracter,file_dest);
			}

}
```

```
void encrypt_CBC(FILE *file, FILE *file_dest){
      head_handler(file,file_dest);

      int caracter=0;
      int **matrix_m;
      int **key;
      int **block;
      creador_matrices_dinamicas(&block,1,3);
      creador_matrices_dinamicas(&matrix_m,1,3);
      creador_matrices_dinamicas(&key,3,3);

      //Matrix
      key[0][0]=1;
      key[0][1]=2;
      key[0][2]=3;
      key[1][0]=4;
      key[1][1]=5;
      key[1][2]=6;
      key[2][0]=11;
      key[2][1]=9;
      key[2][2]=8;

      printf("Insert initial vector: \n");
      scanf("%d %d %d",&block[0][0],&block[0][1],&block[0][2]);
      printf("Initial vector: %d %d %d
\n",block[0][0],block[0][1],block[0][2]);



      while((caracter=fgetc(file))!=EOF){
                  //Read from file
                  matrix_m[0][0]=caracter;
                  matrix_m[0][1]=fgetc(file);
                  matrix_m[0][2]=fgetc(file);
                  //CBC
                  xor(matrix_m,block);
                  hillCipher(matrix_m,key);
                  //Write
                  fputc(matrix_m[0][0],file_dest);
                  fputc(matrix_m[0][1],file_dest);
                  fputc(matrix_m[0][2],file_dest);
                  //Exchange
                  block[0][0]=matrix_m[0][0];
                  block[0][1]=matrix_m[0][1];
                  block[0][2]=matrix_m[0][2];

            }

      }



void encrypt_CFB(FILE *file, FILE *file_dest){
      head_handler(file,file_dest);
```

```
	int caracter=0;
	int **matrix_m;
	int **key;
	int **block;
	creador_matrices_dinamicas(&block,1,3);
	creador_matrices_dinamicas(&matrix_m,1,3);
	creador_matrices_dinamicas(&key,3,3);

	//Matrix
	key[0][0]=1;
	key[0][1]=2;
	key[0][2]=3;
	key[1][0]=4;
	key[1][1]=5;
	key[1][2]=6;
	key[2][0]=11;
	key[2][1]=9;
	key[2][2]=8;

	printf("Insert initial vector: \n");
	scanf("%d %d %d",&block[0][0],&block[0][1],&block[0][2]);
	printf("Initial vector: %d %d %d
\n",block[0][0],block[0][1],block[0][2]);



	while((caracter=fgetc(file))!=EOF){
			//Read from file
			matrix_m[0][0]=caracter;
			matrix_m[0][1]=fgetc(file);
			matrix_m[0][2]=fgetc(file);
			//CFC
			hillCipher(block,key);
			xor(matrix_m,block);
			//Write
			fputc(matrix_m[0][0],file_dest);
			fputc(matrix_m[0][1],file_dest);
			fputc(matrix_m[0][2],file_dest);
			//Exchange
			block[0][0]=matrix_m[0][0];
			block[0][1]=matrix_m[0][1];
			block[0][2]=matrix_m[0][2];

		}

	}


void encrypt_OFB(FILE *file, FILE *file_dest){
	head_handler(file,file_dest);

	int caracter=0;
	int **matrix_m;
	int **key;
	int **block;
	int **aux;
```

```
	creador_matrices_dinamicas(&block,1,3);
	creador_matrices_dinamicas(&matrix_m,1,3);
	creador_matrices_dinamicas(&key,3,3);
	creador_matrices_dinamicas(&aux,1,3);

	//Matrix
	key[0][0]=1;
	key[0][1]=2;
	key[0][2]=3;
	key[1][0]=4;
	key[1][1]=5;
	key[1][2]=6;
	key[2][0]=11;
	key[2][1]=9;
	key[2][2]=8;

	printf("Insert initial vector: \n");
	scanf("%d %d %d",&block[0][0],&block[0][1],&block[0][2]);
	printf("Initial vector: %d %d %d
\n",block[0][0],block[0][1],block[0][2]);



	while((caracter=fgetc(file))!=EOF){
			//Read from file
			matrix_m[0][0]=caracter;
			matrix_m[0][1]=fgetc(file);
			matrix_m[0][2]=fgetc(file);
			//CFC
			hillCipher(block,key);
			aux[0][0]=block[0][0];
			aux[0][1]=block[0][1];
			aux[0][2]=block[0][2];

			xor(matrix_m,block);
			//Write
			fputc(matrix_m[0][0],file_dest);
			fputc(matrix_m[0][1],file_dest);
			fputc(matrix_m[0][2],file_dest);
			//Exchange
			block[0][0]=aux[0][0];
			block[0][1]=aux[0][1];
			block[0][2]=aux[0][2];

		}

	}


void decrypt_CFB(FILE *file, FILE *file_dest){
	head_handler(file,file_dest);
	int caracter=0;
	int **matrix_m;
	int **inv_key;
	int **block;
	int **aux;
	creador_matrices_dinamicas(&block,1,3);
```

```
	creador_matrices_dinamicas(&matrix_m,1,3);
	creador_matrices_dinamicas(&inv_key,3,3);
	creador_matrices_dinamicas(&aux,1,3);

	//Matrix


	inv_key[0][0]=1;
	inv_key[0][1]=2;
	inv_key[0][2]=3;
	inv_key[1][0]=4;
	inv_key[1][1]=5;
	inv_key[1][2]=6;
	inv_key[2][0]=11;
	inv_key[2][1]=9;
	inv_key[2][2]=8;

	//Initial vector
	printf("Insert initial vector: \n");
	scanf("%d %d %d",&block[0][0],&block[0][1],&block[0][2]);
	printf("Initial vector: %d %d %d
\n",block[0][0],block[0][1],block[0][2]);

	while((caracter=fgetc(file))!=EOF){

			matrix_m[0][0]=(unsigned char)caracter;
			matrix_m[0][1]=(unsigned char)fgetc(file);
			matrix_m[0][2]=(unsigned char)fgetc(file);

			aux[0][0]=matrix_m[0][0];
			aux[0][1]=matrix_m[0][1];
			aux[0][2]=matrix_m[0][2];

			//Decrypt
			hillCipher(block,inv_key);
			//XOR
			xor(matrix_m,block);

			fputc(matrix_m[0][0],file_dest);
			fputc(matrix_m[0][1],file_dest);
			fputc(matrix_m[0][2],file_dest);
			//Aux
			block[0][0]=aux[0][0];
			block[0][1]=aux[0][1];
			block[0][2]=aux[0][2];



		}
	}

void decrypt_CBC(FILE *file, FILE *file_dest){
	head_handler(file,file_dest);
	int caracter=0;
	int **matrix_m;
	int **inv_key;
	int **block;
```

```
	int **aux;
	creador_matrices_dinamicas(&block,1,3);
	creador_matrices_dinamicas(&matrix_m,1,3);
	creador_matrices_dinamicas(&inv_key,3,3);
	creador_matrices_dinamicas(&aux,1,3);

	//Matrix
	inv_key[0][0]=90;
	inv_key[0][1]=167;
	inv_key[0][2]=1;
	inv_key[1][0]=74;
	inv_key[1][1]=179;
	inv_key[1][2]=254;
	inv_key[2][0]=177;
	inv_key[2][1]=81;
	inv_key[2][2]=1;

	//Initial vector
	printf("Insert initial vector: \n");
	scanf("%d %d %d",&block[0][0],&block[0][1],&block[0][2]);
	printf("Initial vector: %d %d %d
\n",block[0][0],block[0][1],block[0][2]);

	while((caracter=fgetc(file))!=EOF){

			matrix_m[0][0]=(unsigned char)caracter;
			matrix_m[0][1]=(unsigned char)fgetc(file);
			matrix_m[0][2]=(unsigned char)fgetc(file);

			aux[0][0]=matrix_m[0][0];
			aux[0][1]=matrix_m[0][1];
			aux[0][2]=matrix_m[0][2];

			//Decrypt
			hillCipher(matrix_m,inv_key);
			//XOR
			xor(matrix_m,block);

			fputc(matrix_m[0][0],file_dest);
			fputc(matrix_m[0][1],file_dest);
			fputc(matrix_m[0][2],file_dest);
			//Aux
			block[0][0]=aux[0][0];
			block[0][1]=aux[0][1];
			block[0][2]=aux[0][2];



		}
	}

void decrypt_OFB(FILE *file, FILE *file_dest){
	head_handler(file,file_dest);
	int caracter=0;
	int **matrix_m;
	int **inv_key;
	int **block;
```

```
	int **aux;
	creador_matrices_dinamicas(&block,1,3);
	creador_matrices_dinamicas(&matrix_m,1,3);
	creador_matrices_dinamicas(&inv_key,3,3);
	creador_matrices_dinamicas(&aux,1,3);

	//Matrix

	inv_key[0][0]=1;
	inv_key[0][1]=2;
	inv_key[0][2]=3;
	inv_key[1][0]=4;
	inv_key[1][1]=5;
	inv_key[1][2]=6;
	inv_key[2][0]=11;
	inv_key[2][1]=9;
	inv_key[2][2]=8;

	//Initial vector
	printf("Insert initial vector: \n");
	scanf("%d %d %d",&block[0][0],&block[0][1],&block[0][2]);
	printf("Initial vector: %d %d %d
\n",block[0][0],block[0][1],block[0][2]);

	while((caracter=fgetc(file))!=EOF){
			//Decrypt
			hillCipher(block,inv_key);

			matrix_m[0][0]=(unsigned char)caracter;
			matrix_m[0][1]=(unsigned char)fgetc(file);
			matrix_m[0][2]=(unsigned char)fgetc(file);

			//XOR
			xor(matrix_m,block);

			fputc(matrix_m[0][0],file_dest);
			fputc(matrix_m[0][1],file_dest);
			fputc(matrix_m[0][2],file_dest);



		}
	}




void xor(int **matrix_m,int **block){
	//printf("Block a: %d %d %d. Block b: %d %d
%d.\n",matrix_m[0][0],matrix_m[0][1],matrix_m[0][2],block[0][0],block[0][
1],block[0][2]);
	matrix_m[0][0]^=block[0][0];
	matrix_m[0][1]^=block[0][1];
	matrix_m[0][2]^=block[0][2];
	//printf("XORed block %d %d %d
\n",matrix_m[0][0],matrix_m[0][1],matrix_m[0][2]);
}
```

Seal of approval