INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO

Cryptography

Affine Cipher

Abstract

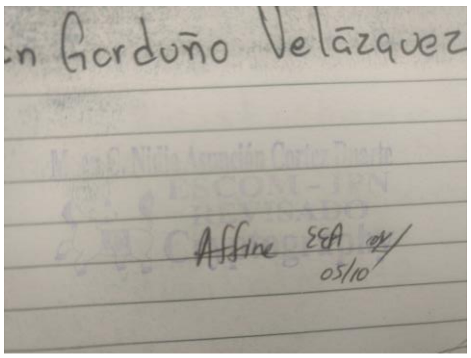
Affine cipher which is a classical cipher, easy to implement and easy to use. In this document I will describe the mathematical fundaments of Affine cipher, and show an implementation following that fundaments.

By:

Garduño Velazquez Alan

Professor:
MSc. NIDIA ASUNCIÓN CORTEZ DUARTE

October 2017


n Garduño Velázquez
M. en C. Nidia Asunción Cortez Duarte
ESCOM - IPN
REVISADO
Affine EEA
05/10

Index

Introduction:

Hide the information always has been so important, since long time ago the kings, emperors, politicians have been looking a way to hide their private messages, plans or simply sensitive information to protect the information from non-desired looks. This research of a way to hide the information, results on the develop of different techniques to hide it, for example the Caesar cipher that makes a 3 shift on the letter, so if you have “a” in the original message and then applies Caesar cipher you obtain a “D”, at first this cipher was unbreakable because the people don’t know how it works nut truly it was very simple and very easy to break it, so quickly that cipher no use any more. Time after in the history we have one of the most beautiful machines that the man has created Enigma Machine, this machine was used to cipher the Nazi messages, attack plans and strategies. when Alan Turing and his team found how to decrypt the messages that the machine generates probably changed the course of the history, that fact allows the Allies to know how to attack and win the war, So Could you imagine what have would happened if anyone have not would found how to break the Enigma machine? That is the impact that the cryptography has in our history.

In this document we are going to show how the affine cipher works, the theory and its mathematical fundaments explained, also we are going to show a basic implementation of this cipher following the theory and the mathematical fundaments showing the results and explain them showing the chart flow and the code of the implementation in C language.

Literature review:

Affine Cipher needs two values to work alpha (α) and beta (β), being alpha a multiplicative value and beta the additive value, but is not that simple alpha and the ring (alphabet size) must be co-primes each other to correct work of the cipher, this means that the gcd (greatest common divisor) must be one, that can be probed using Euclides algorithm. (See the examples below)

Example 1.

$\gcd(11,26) = 1$

$26 = 11(2) + 4$

$11 = 4(2) + 3$

$4 = 3(1) + 1$

This number will work perfect because gcd between the numbers is equals to one.

Example 2.

$\gcd(26,13) = 13$

26 = 13(2) + 0

This number won´t work because the gcd between the numbers is different of one.

Then it's necessary to give to every letter of the alphabet a value, in this case we are using the English alphabet so the letter 'a' will take the value of 0, 'b' of 1, c=3, …. , z=25, as we can see we just have 26 values so all the operations we will do need apply mod 26, and just like we said before alpha and alphabet size (in this case 26) must be co-primes, finally the formula to encrypt with the affine cipher is given below

$$E_k = (\propto p + \beta) \text{ mod } 26$$

Where E_k is the function, $\propto$ is the alpha value, β is the beta value and p is the value of the letter in plain.

It means, that each letter of the message, we need to multiply by alpha's values, then add beta's values and finally, applying module alphabet's size (in this case is 26), it gives us the encrypted message, for convention the encrypted message must be in Capital Letters.

We just show how to encrypt, but an important part of a good cipher decrypt, so now we are going to explain how to decrypt.

For decrypt we need again the alpha and beta, well no at all, that we really need is the inverse multiplicative and the inverse additive of alpha and beta respectively.

Both are easy to calculate, for the inverse multiplicative we will need the Euclides algorithm but in this time whit its extension, we already know that alpha and the alphabet size must be co-primes, to probe that we use Euclides Algorithm, so now we are going to do some more steps to obtain the inverse multiplicative.

View Example 1, in the example 1 we did the Euclides algorithm, so now, here is how to do the extension.

E.E.A. (Eculides Extended Algorithm) for the Example 1

Well, first we need the Euclides algorithm, we already do it in the example 1, so to the extension we need to clear the last number

gcd(11,26) = 1

26 = 11(2) + 4-------------------------------4 = 26 -11(2) ……… (c)

11 = 4(2) + 3---------------------------------3 = 11 - 4(2) ………. (b)

4 = 3(1) + 1----------------------------------1 = 4 - 3(1) ………… (a)

Now we have 3 new equations, we must substitute the equation (b) in the equation (a) and the (c) in (b)

$1 = 4 - [11 - 4(2)] (1)$

$1 = 4 -11 + 4(2)$

$1 = 4(3) – 11$

$1 = [26 – 11(2)] (3) – 11$

$1 =26(3) – 11(6) – 11$

1 = **26(3) – 11(7)**

This means that the inverse multiplicative of 11 is 7.

To calculate the inverse additive of beta we have 2 cases, if $\beta < 26$ we will have to add the number to satisfy the equation $\beta + (-n) = 26$ where n is the inverse additive, in the case if $\beta > 26$, first we have to do βmod26 and the repeat until $\beta < 26$ then add the number to satisfy the equation that was before.

Finally, we have the formula to decrypt:

$$D_k = \propto^{-1} (C + \beta^{-1})\, mod\, 26$$

Where D_k is the function, $\propto^{-1}$ is the inverse multiplicative of alpha value, β^{-1} is the inverse additive of beta value and C is the value of the letter ciphered.

Software (libraries, packages, tools):

Libraries[1]:

To develop this program, I used the standard C libraries:

<stdio.h>

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

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

And Macro EOF

<stdlib.h>

For manege dinamic memory , the used function

- Malloc()

<string,h>

For obtain the length of a string, the used function was:

- Strlen**();**

Tools:

- Sublime text 3
- Visual Paradigm 14.2
- Clion Debugger

Procedure:

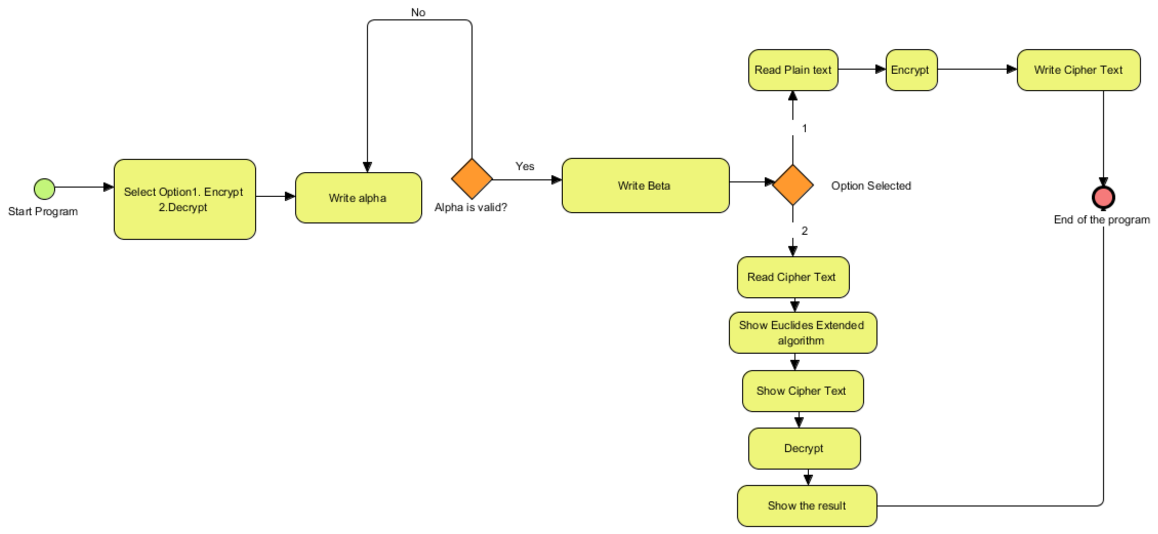


Figure 1. Affine cipher chart flow

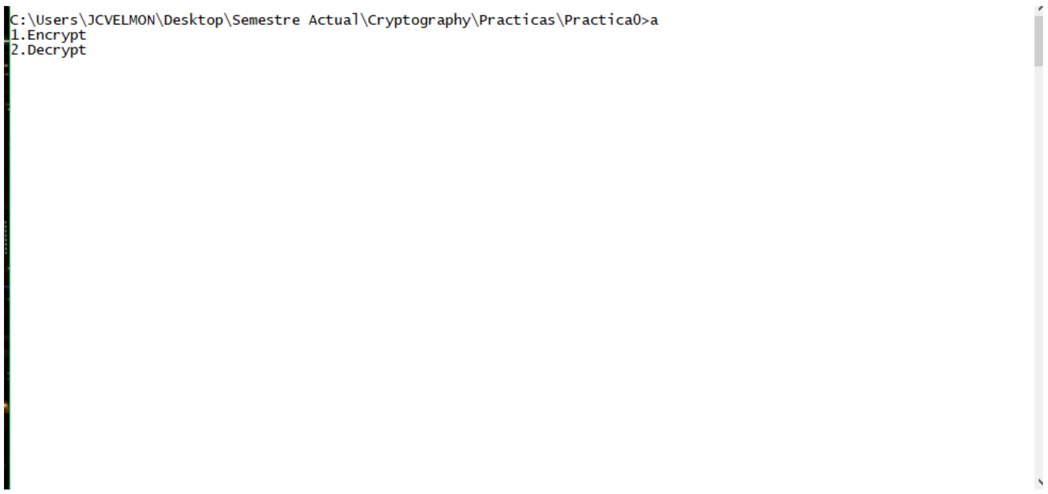
First, we need to create two files “m.txt” that one will contains the plain text and “c.txt” in that one we will write the cipher text.

After that we will ask to user what does he want to do? Encrypt or Decrypt, the first time “c.txt” will be empty so, you need to encrypt the content of m.txt, then we will ask for the values of alpha and beta, the program validates that the number and the alphabet size (in this case 26) meet the condition mentioned in the literature review so gcd(alpha, 26) must be equals to 1, if not the program will ask for a new value until the value meet the condition, the we will ask for the beta value and the we will show the content of m.txt, in the background we read character by character and apply the affine cipher formula to encrypt, to decrypt we made exactly the same but in this case we use the affine decipher formula.

In a general description:

1. Ask to the user what does he want to do?
2. Ask for a valid value of alpha and beta.
3. Read character by character and apply the correspondent formula.
4. Show the results.

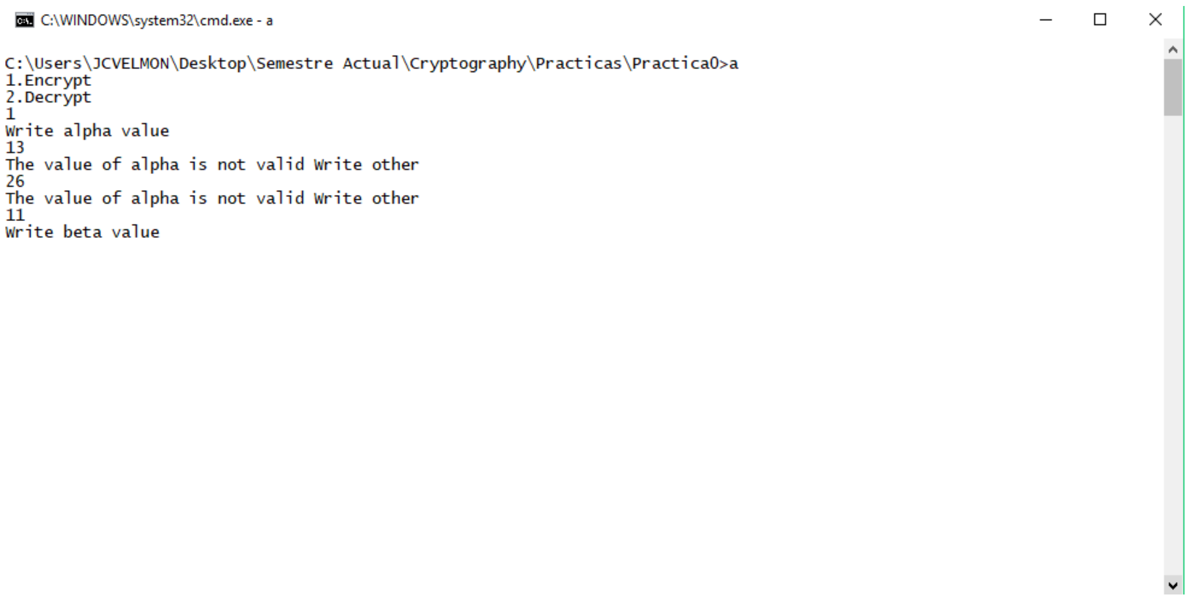
Results



```
C:\Users\JCVELMON\Desktop\Semestre Actual\Cryptography\Practicas\Practica0>a
1.Encrypt
2.Decrypt
```

Figure 1. Main menu of the program selecting option 1 (Encrypt).

In figure 2, We can see the main menu, selecting first the option if you want to encrypt or decrypt a message from a file, then, you must put the values that alpha and beta are going to have through the execution of the program to obtain the cipher.

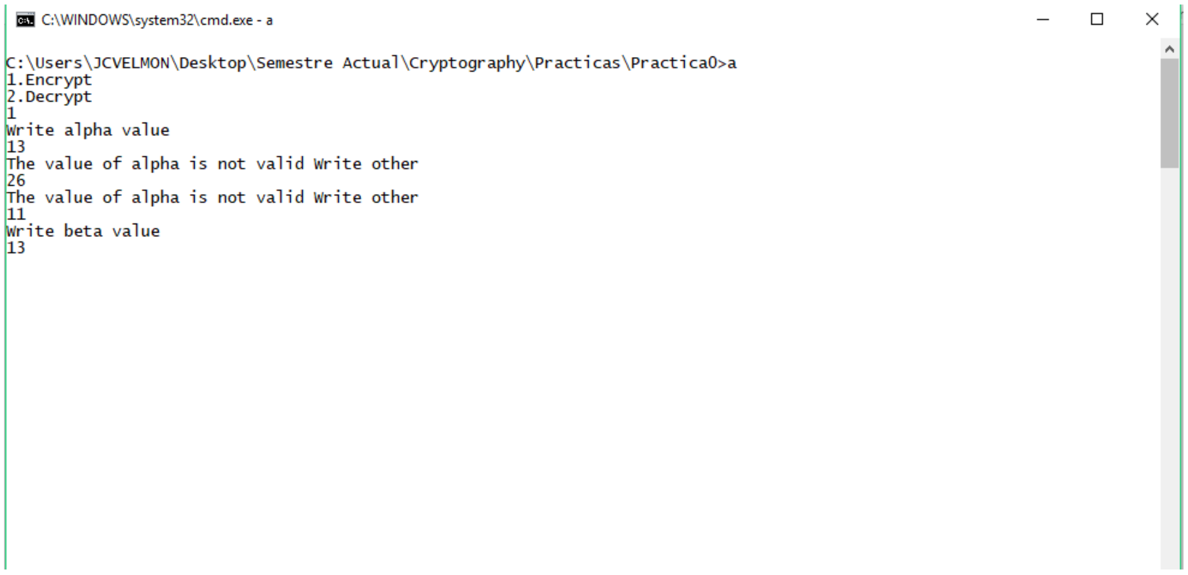

```
C:\WINDOWS\system32\cmd.exe - a

C:\Users\JCVELMON\Desktop\Semestre Actual\Cryptography\Practicas\Practica0>a
1.Encrypt
2.Decrypt
1
Write alpha value
13
The value of alpha is not valid Write other
26
The value of alpha is not valid Write other
11
Write beta value
```

Figure 2. Validating alpha

In figure 3 we intentionally put a wrong value of alpha to see how the program ask for another one until the value is valid and the ask for the beta value (View Figure 3.)

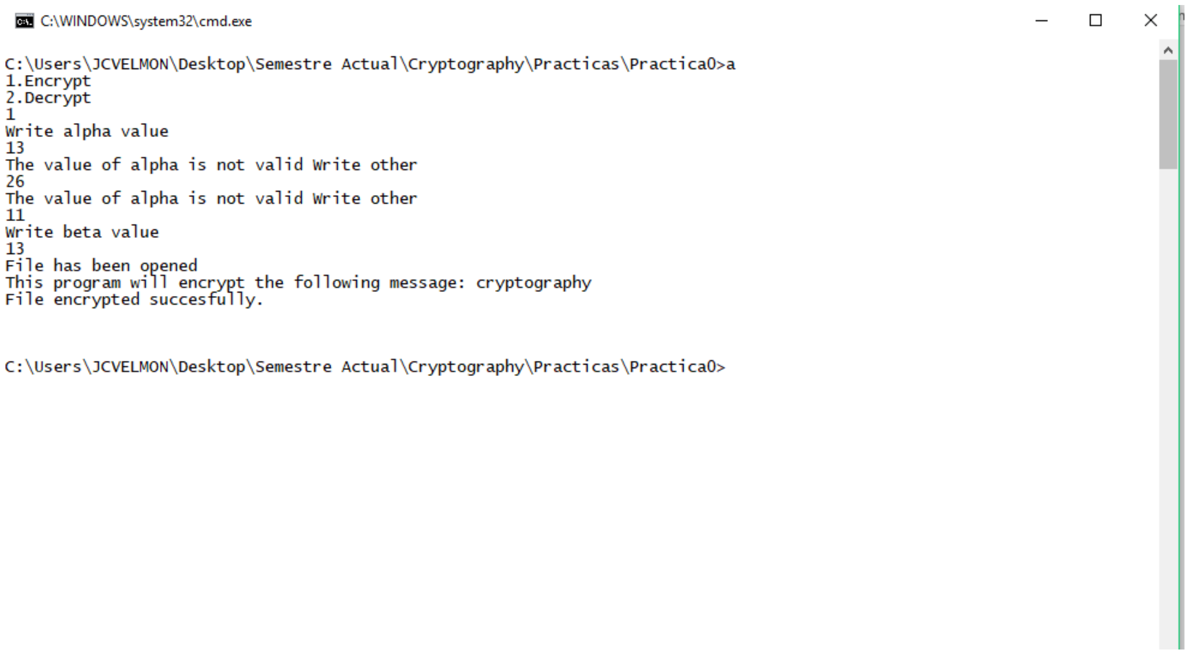
To demonstrate functionality of the program we will use values of alpha that we used in the example 1 (11) and for the beta value we will use 13. (View Figure 4.)


```
C:\WINDOWS\system32\cmd.exe - a

C:\Users\JCVELMON\Desktop\Semestre Actual\Cryptography\Practicas\Practica0>a
1.Encrypt
2.Decrypt
1
Write alpha value
13
The value of alpha is not valid Write other
26
The value of alpha is not valid Write other
11
Write beta value
13
```

Figure 4. Typing alpha and beta values

Then we show the plain text and a confirmation of the encryption. (View Figure 5)



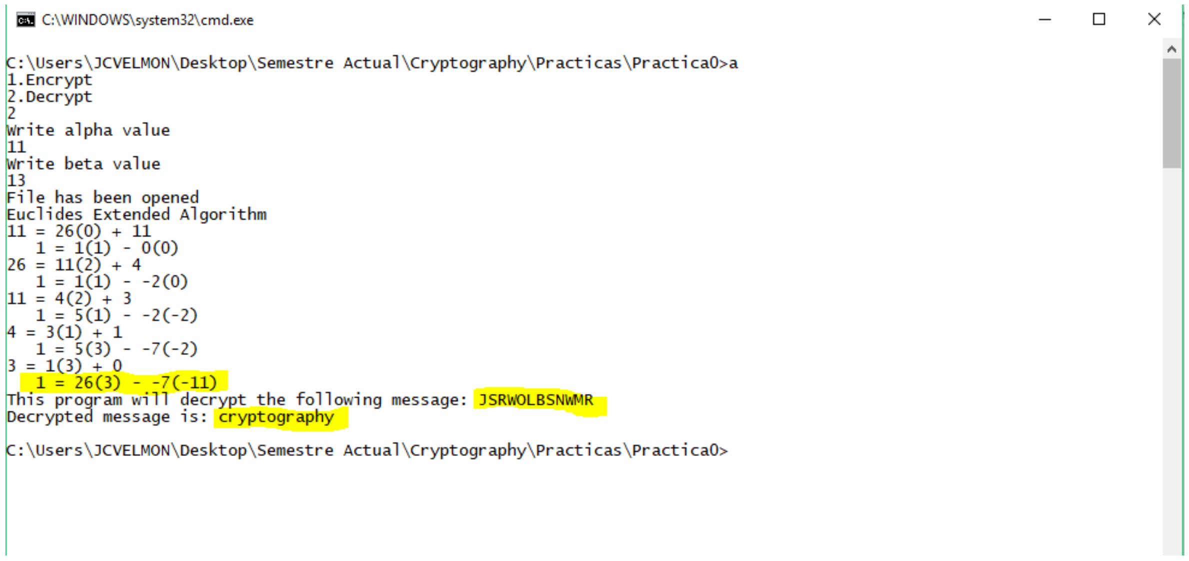
```
C:\WINDOWS\system32\cmd.exe

C:\Users\JCVELMON\Desktop\Semestre Actual\Cryptography\Practicas\Practica0>a
1.Encrypt
2.Decrypt
1
Write alpha value
13
The value of alpha is not valid Write other
26
The value of alpha is not valid Write other
11
Write beta value
13
File has been opened
This program will encrypt the following message: cryptography
File encrypted succesfully.


C:\Users\JCVELMON\Desktop\Semestre Actual\Cryptography\Practicas\Practica0>
```

Figure 5. Showing plain text and confirmation message.

Now we will select decryption option and we type the values of alpha and beta, after that the program will show the E.E.A[2], the cipher text and the plain text. (View Figure 6).



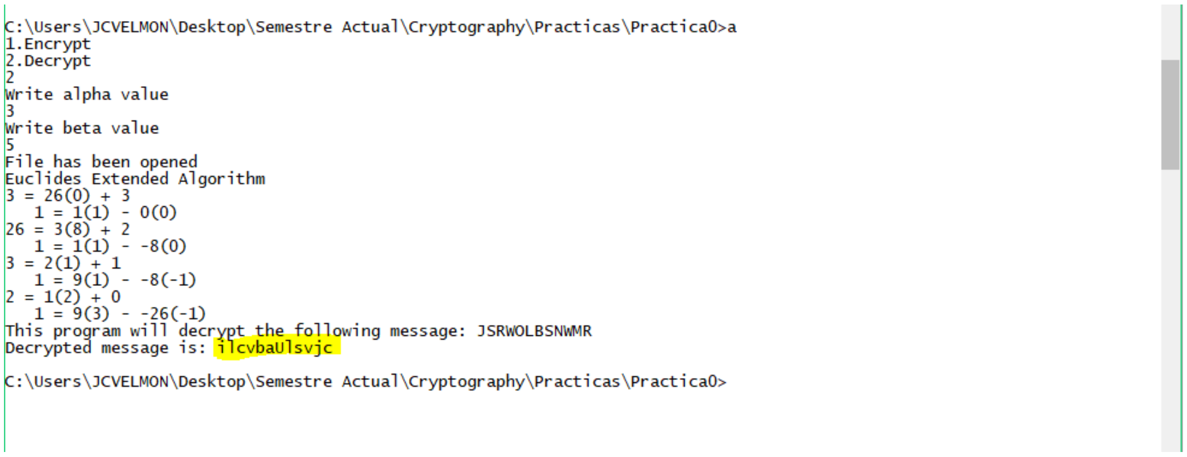
```
C:\WINDOWS\system32\cmd.exe

C:\Users\JCVELMON\Desktop\Semestre Actual\Cryptography\Practicas\Practica0>a
1.Encrypt
2.Decrypt
2
Write alpha value
11
Write beta value
13
File has been opened
Euclides Extended Algorithm
11 = 26(0) + 11
   1 = 1(1) - 0(0)
26 = 11(2) + 4
   1 = 1(1) - -2(0)
11 = 4(2) + 3
   1 = 5(1) - -2(-2)
4 = 3(1) + 1
   1 = 5(3) - -7(-2)
3 = 1(3) + 0
   1 = 26(3) - -7(-11)
This program will decrypt the following message: JSRWOLBSNWMR
Decrypted message is: cryptography

C:\Users\JCVELMON\Desktop\Semestre Actual\Cryptography\Practicas\Practica0>
```

Figure 6. Decryption process.

So what happens if we put and incorrect values of alpha and beta, the program return an unreadable text. (View Figure 7).



```
C:\Users\JCVELMON\Desktop\Semestre Actual\Cryptography\Practicas\Practica0>a
1.Encrypt
2.Decrypt
2
Write alpha value
3
Write beta value
5
File has been opened
Euclides Extended Algorithm
3 = 26(0) + 3
   1 = 1(1) - 0(0)
26 = 3(8) + 2
   1 = 1(1) - -8(0)
3 = 2(1) + 1
   1 = 9(1) - -8(-1)
2 = 1(2) + 0
   1 = 9(3) - -26(-1)
This program will decrypt the following message: JSRWOLBSNWMR
Decrypted message is: ilcvbaUlsvjc

C:\Users\JCVELMON\Desktop\Semestre Actual\Cryptography\Practicas\Practica0>
```

*F*igure 7 Decryption process with incorrect values of alpha and beta.

Discussion:

As we can see in the results the cipher and decipher works well, we use the alpha value of the example 1 to probe that our program uses correctly the modular arithmetic, so if you see the Figure 6 (in results) and the procedure that we do in the Example 1 and the extension (In Literature review but also below).

The result is the same, so our program uses correctly the algorithm and the modular arithmetic.

Example 1 and E.E.A (explained in Literature Review)

gcd(11,26) = 1

26 = 11(2) + 4

11 = 4(2) + 3

4 = 3(1) + 1

26 = 11(2) + 4---------------------------------4 = 26 -11(2) ……… (c)

11 = 4(2) + 3-----------------------------------3 = 11 - 4(2) ………. (b)

4 = 3(1) + 1------------------------------------1 = 4 - 3(1) ………… (a)

1 = 4 - [11 - 4(2)] (1)

1 = 4 -11 + 4(2)

1 = 4(3) – 11

1 = [26 – 11(2)] (3) – 11

1 =26(3) – 11(6) – 11

1 = **26(3) – 11(7)** **(View Figure 6{in results} to do a comparison between results)**

As you can see the theory is so important to understand apply the correct process and then when you get your results, them should be clear and everyone can understand them.

Conclusions:

I learned how to follow a mathematical fundament and apply them in a program that has a real application in this case a classic cipher, I think I never before did something similar I used to use only brute force and not use the theory fundaments that supports it.

One of the errors happens when de alpha has a too longer value, is because the gdc algorithm that I use is a recursive algorithm so if alpha is too big (its start to crash when alpha > 999999999999) the program crash. Another one is when the plain text is too large (more than 200 characters) the program is unable to read the file and crash.

Generally, this program works with 26 alphabet size but could be n whit a little modifications, as well it works whit any alpha and beta values minors of 999999999999, is a big number so in a normal use can work perfectly.

This program, is far to be a definitive version, it needs a lot of optimization in performance and some extra validation, so this procedure can be optimized an improved a lot.

References:

1. CplusCplus, “Library Reference”, 2000, [Online] Available on:
http://www.cplusplus.com/reference/

2. Andrés Esquivel, ‘IMPLEMENTACIÓN DEL ALGORITMO DE EUCLIDES EXTENDIDO EN JAVA’, 2015, [Online]. Available on:
http://blog.andresed.me/2015/06/implementacion-de-euclides-extendido-en.html.

3. Examples and their explication and the formulas was taken from Cryptography aim, ESCOM,2017 Prof.: Nidia Asunción Cortez Duarte.

4. Programiz,” C Program to Find G.C.D Using Recursion”, [Online] Available on:

https://www.programiz.com/c-programming/examples/hcf-recursion

**All the online references are available on 20/10/17

Code

Main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "cipher.c"

int main(void)
{
    int op;
    int alpha,beta;
    printf("1.Encrypt\n2.Decrypt\n");
    scanf("%d",&op);
    printf("Write alpha value\n");
    scanf("%d",&alpha);
    if(gcd(alpha,26) != 1){
        while(gcd(alpha,26) != 1){
            printf("The value of alpha is not valid Write other\n");
            scanf("%d",&alpha);
        }
    }
    printf("Write beta value\n");
    scanf("%d",&beta);

    switch(op){
        case 1:
            encrypt(alpha,beta);
        break;
        case 2:
            decrypt(alpha,beta);
        break;

    }
    return 0;
}
```

Cipher.h

```c
char * read(char mode);
void write(char * text);
void encrypt(int alpha, int beta);
void decrypt(int alpha, int beta);
int gcd(int alpha, int alphabet);
int alg_euc_ext(int n1, int n2);
int invAd(int beta);
```

Cipher.c

Note: The code of the function `int alg_euc_ext(int n1, int n2);` was taken from the reference 2. And the function `int gcd(int alpha, int alphabet);` was taken from the reference 4.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "cipher.h"
```

```c


//Code reference (2)
int alg_euc_ext(int n1,int n2) {
    int array[3],x=0,y=0,d=0,x2 = 1,x1 = 0,y2 = 0,y1 = 1,q = 0, r = 0;
    int flag=1;
        int in=n1;
    if(n2==0){
        array[0]=n1;
        array[1]=1;
        array[2]=0;
    }
    else{
        while(n2>0){
            q = (n1/n2);
            r = n1 - q*n2;
            x = x2-q*x1;
            y = y2 - q*y1;
            n1 = n2;
            n2 = r;
            x2 = x1;
            x1 = x;
            y2 = y1;
            y1 = y;
    if(flag%2 != 0){
        printf("%d = %d(%d) + %d        \n   1 = %d(%d) - %d(%d)  \n",n1*q+r,n1,q,r,x1,y2,x2,y1 );
    }else{
        printf("%d = %d(%d) + %d        \n   1 = %d(%d) - %d(%d)  \n",n1*q+r,n1,q,r,x2,y1,x1,y2 );
        }
        flag++;
    }
        array[0] = n1;   // mcd (n1,n2)
        array[1] = x2;   // x
        array[2] = y2;   // y
    }

    return array[2]; //Alfa

}

int invAd(int beta){
    int tmp = 0;
    while(beta < 26){
        tmp++;
        beta++;
    }
        if((tmp%26) == 0)
            printf("%d\n",tmp );
            return tmp;
    printf("%d\n",tmp );
    return tmp;
}

//Code reference (4)
int gcd (int alpha, int alphabet)
{
    int temp, inverse, x;
    if (alpha > alphabet)
    {
        temp = alphabet;
        alphabet = alpha;
        alpha = temp;
    }
    if (alpha != 0)
    {

        gcd (alpha, alphabet % alpha);
    }else
    {
        return alphabet;
    }
}

void encrypt(int alpha, int beta){
    int i,c;
    char *plainText = (char*)malloc(sizeof(char));
    char *cipherText = (char*)malloc(sizeof(char));
    plainText = read('e');
    printf("This program will encrypt the following message: %s\n",plainText );
    for (i = 0; i <strlen(plainText);i++)
    {
        c = plainText[i] - 97;
        c = c*alpha;
        c = c + beta;
        c = c%26;
        cipherText[i] = c + 65; //To upper case
    }
    cipherText[i] = '\0';
    write(cipherText);
}
void decrypt(int alpha, int beta){
    int i,c, inverse, additive;
    char *plainText = (char*)malloc(sizeof(char));
    char *cipherText = (char*)malloc(sizeof(char));
    cipherText = read('d');
          printf("Euclides Extended Algorithm\n");
          inverse = alg_euc_ext(alpha,26);
          additive = invAd(beta);
          printf("This program will decrypt the following message: %s\n",cipherText);
          for (i = 0; i < strlen(cipherText); ++i)
          {
              c = cipherText[i] - 65;
              c = c * inverse;
              c = c - additive;
              c = c % 26;
              plainText[i] = c + 97; //to lower case
          }
          plainText[i] = '\0';
          printf("Decrypted message is: %s\n",plainText);



      }

      char * read(char mode){
          FILE *file;
          int i = 0;
          char c, *texInFile = (char*)malloc(sizeof(char));
          if(mode == 'e'){
              file = fopen("m.txt","r");
              if(file == NULL){
                  printf("Unable to read the m.txt file\n");
                  exit(0);
              }
              else
                  printf("File has been opened \n");
              c = fgetc(file);
              while(c != EOF)
              {
                  if (c != 32 && c != '\n')
                  {
                      if ((c >= 'a' && c <= 'z'))
                          texInFile [i ++] = c;
                      else
                      {
                      printf("File m.txt has to contain only lower case characters, otherwise the cipher won't work wwell\n");
                      exit (0);
                      }
                  }
                  c = fgetc(file);
              }

              texInFile[i] = '\0'; // Null character
              fclose(file);
              return texInFile;

          }
          else{
              file = fopen("c.txt","r");
              if(file == NULL){
                  printf("Unable to read the c.txt file\n");
                  exit(0);
              }
              else
                  printf("File has been opened \n");
              c = fgetc(file);
              while(c != EOF){
                  if (c != 32 && c != '\n')
                  {
                      if ((c >= 'A' && c <= 'Z'))
                          texInFile [i ++] = c;
                      else
                      {
                      printf("File c.txt has to contain only upper case characters, otherwise the decipher won't work wwell\n");
                      exit (0);
                      }
                  }
                  c = fgetc(file);
              }
              texInFile[i] = '\0'; // Null character
              fclose(file);
              return texInFile;

          }
      }

      void write(char * text){
          FILE * file;
          file = fopen ("c.txt", "w");
          if (file == NULL){
              printf("Error opening: 'c.txt'\n");
              exit(0);
          }
          fprintf(file, "%s", text);
          printf("File encrypted succesfully.\n");
          fclose (file);
      }
```