



INSTITUTO POLITÉCNICO NACIONAL ESCUELA SUPERIOR DE CÓMPUTO



Cryptography

Affine Cipher

Abstract

Affine is one of the most popular classical ciphers all over the world, it consists of 2 parameters called alpha and beta, to multiply and add respectively each value of each letter of the original message to replace each letter with another with some restrictions explained on this practice.

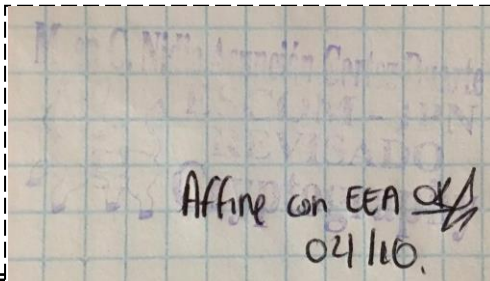
By:

Romero Gamarra Joel Mauricio

Professor:

MSc. NIDIA ASUNCIÓN CORTEZ DUARTE

October 2017



Index

Content

Introduction:	1
Literature review:	1
Software (libraries, packages, tools):.....	2
Procedure:	3
Results	3
Discussion:	7
Conclusions:	7
References:	8
Code	9

Introduction:

Since long time ago, there was a necessity, hide messages to prevent that some people you don't want to, read your message and know what you said on it. The first ciphers in the world date back to the V Century B.C, it consists on a wooden stick, with a leather band rolled through the stick with the message write it on it and then unroll the band and write letter with no sense. The **key** for this cipher was **the diameter** of the wooden stick, cause with another, the message wouldn't be clear.¹

Another famous cipher is César's, by Romans, it consists on having the message clear, and then substitute the letter with the letter 3 positions to the right, for example: a = d, b = e, ..., z = c. The **key** for this cipher is **shift 3 positions to the right**.¹

There are some classical ciphers throughout the world that are so famous, one of those is Affine cipher, it has basically the same idea that César's cipher, with the only difference that it has another parameter (multiplicative) and another difference is that in César cipher, the shift was always 3 positions to the right, here, the shift could be 2, 3, 4, ..., (alphabet's size - 1). The **key** for this cipher is the 2 parameters, that we'll call **alpha and beta**.

Literature review:

Affine Cipher, one of the most famous classical ciphers through history, it needs 2 values to encrypt the message, alpha (multiplicative value) and beta (additive value).

Either alpha or beta, they must have a value between 1 and alphabet size (in this case, English alphabet has 26 letters). In addition, alpha and the alphabet size must be prime to each other to obtain a great result encrypting the message, this means, that the greatest common divisor (GCD) between alpha and 26 should be 1 to guarantee they're prime each other.

Affine cipher consists first, at giving to each letter of the alphabet a value, for example a = 0, b = 1, c = 2, ..., z = 25. Then, the formula for encrypt a message with Affine cipher, is the following:

$$E_k = (\alpha p + \beta) \bmod 26$$

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 (in the successive, with capital letters) and we can decrypt it by founding the inverse additive and the inverse multiplicative for alpha and beta. The formula for decrypt a message is the following:

$$D_k = \alpha^{-1} (C + \beta^{-1}) \bmod 26$$

Inverse additive is too simple, we only need to know that $26 \bmod 26 = 0$, so, what we need to do next is founding a number that in addition to β is 26. However, the multiplicative inverse, is a little more complicated to calculate, but not too much, we only need to remember that $\alpha \cdot \alpha^{-1} \bmod 26 = 1$.

One of the most useful algorithms in this practice is Euclides's algorithm², that consists on founding de greatest common divisor between 2 numbers in a fast way. Computationally, we can apply modular division between both numbers, and then use a temporal variable to save the previous number, breaking the cycle when the result of the modular division is zero and then the result will be the previous number before that zero. It is too useful, due to its spatial complexity and temporal complexity, reducing execution time, used memory and useless operations, giving the result faster and permitting creating new applications for the algorithm based on it.²

Software (libraries, packages, tools):

Libraries:³

- Stdio.h: Used for the following functions:
 - `int printf(const char *format, ...)`
 - `int scanf(const char *format, ...)`
 - `FILE *fopen(const char *filename, const char *mode)`
 - `int fgetc(FILE *stream)`
 - `int feof(FILE *stream)`
 - `int fprintf(FILE *stream, const char *format, ...)`
 - `int fclose(FILE *stream)`
- Stdlib.h: Used for the following functions:
 - `void *malloc(size_t size)`
 - `void free(void *ptr)`
 - `void exit(int status)`
 - `int system(const char *string)`
- String.h: Used for the following functions:
 - `size_t strlen(const char *str)`
- Functions.h (Own): Used the following functions, making use of the above functions:
 - `void encrypt (int alpha, int beta)`
 - `char * readMessage ()`
 - `void writeCiphertext (char * ciphertext)`
 - `void decrypt (int alpha, int beta)`
 - `char * readCiphertext ()`
 - `int multiplicativeInverse (int alpha)`
 - `void menu ()`
 - `void validateNumbers (int alpha, int beta)`
 - `int gcd (int alpha, int alphabet)` ← Euclides's algorithm

Tools:

- Sublime text 3
- Bizagi Modeler

Procedure:

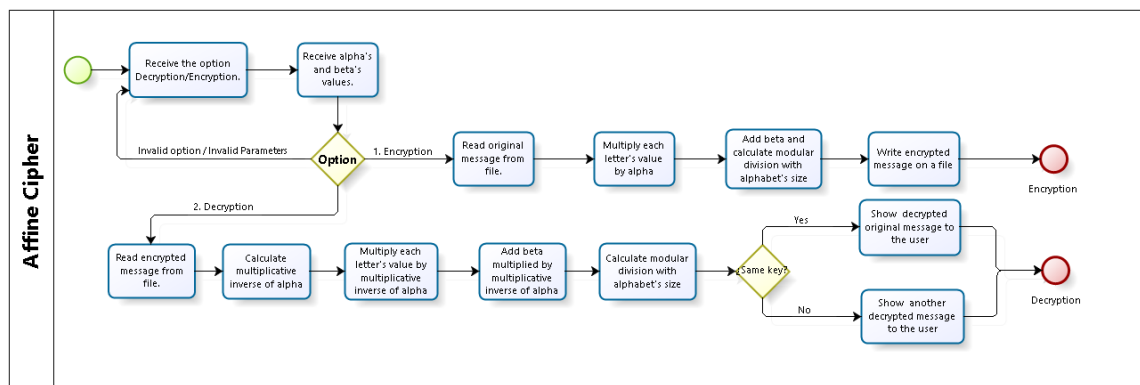


Figure 1. Affine Cipher Flowchart

Affine cipher works the following way:

- Assign a value to each letter. $a = 0, b = 1, c = 2, d = 3, e = 4, \dots, y = 24, z = 25$.
- Multiply each value by alpha (given by the user), for example, $\alpha = 5$. $c = 2(3) = 6$.
- Add to the new value, the value of beta, for example, $\beta = 14$. $c = 6 + 14 = 20$.
- Apply modular division to the new value with alphabet's size. $c = 20 \% 26 = 20$.
- The new letter is given by this last operation, in this example with those parameters, the letter encrypted that corresponds to the original letter 'c' is letter 'U' in capital letter only to distinguish the encrypted and original messages.

What I did, is re-create the steps explained above, with some functions in C language and then substitute the letter with the new one and write it on a message, with the purpose to obtain the decrypted message (with the correct key) at any moment the user wants, reading the encrypted message and introducing the values for alpha and beta to discover the ciphered text.

Results

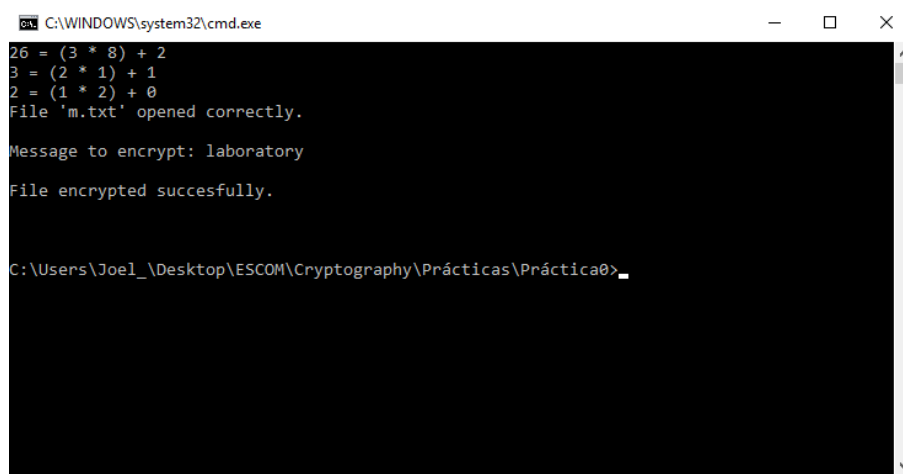
```
C:\WINDOWS\system32\cmd.exe - Affine.exe

Would you like to encrypt or decrypt?
1. Encrypt
2. Decrypt
1

Alfa's value: 3
Beta's value: 15_
```

Figure 2. Main menu of the program selecting option 1 (Encrypt).

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



```
C:\WINDOWS\system32\cmd.exe
26 = (3 * 8) + 2
3 = (2 * 1) + 1
2 = (1 * 2) + 0
File 'm.txt' opened correctly.
Message to encrypt: laboratory
File encrypted succesfully.

C:\Users\Joel_\Desktop\ESCOM\Cryptography\Prácticas\Práctica0>
```

Figure 3. Execution of encryption showing the message, a text if the ciphered was successful and the steps corresponding to the Euclidean algorithm.

After we receive the message that the encryption was successful, we'll proceed to review the text to check if it's correctly ciphered (Figure 4) and compared to a web page⁴ with the same values and the same message (Figure 5).

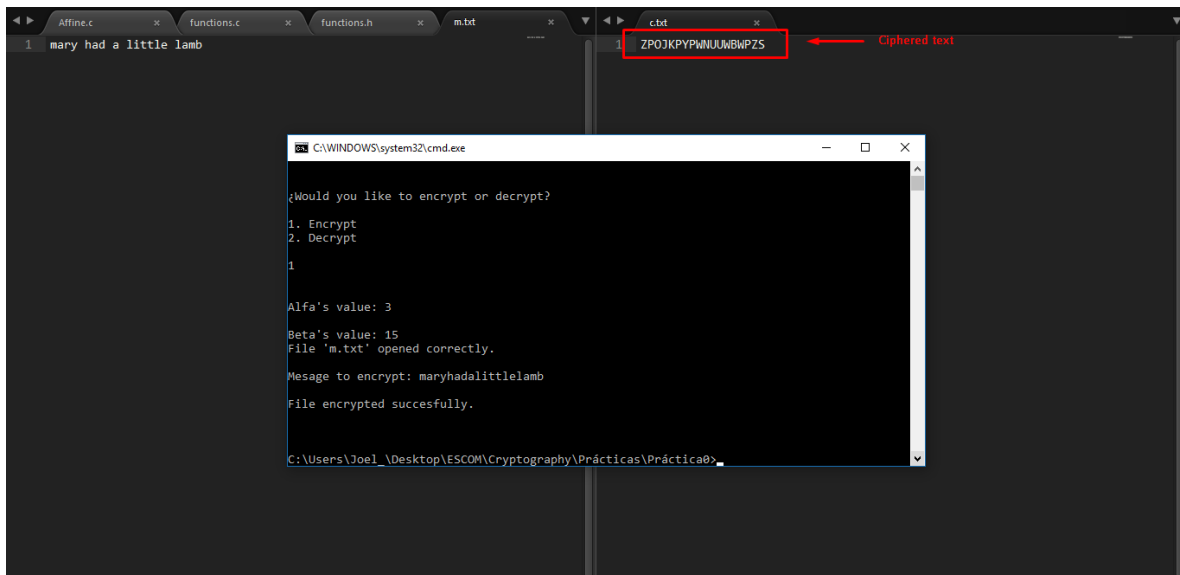


Figure 4. Demonstration of the encrypted message (right) with the original one (left).

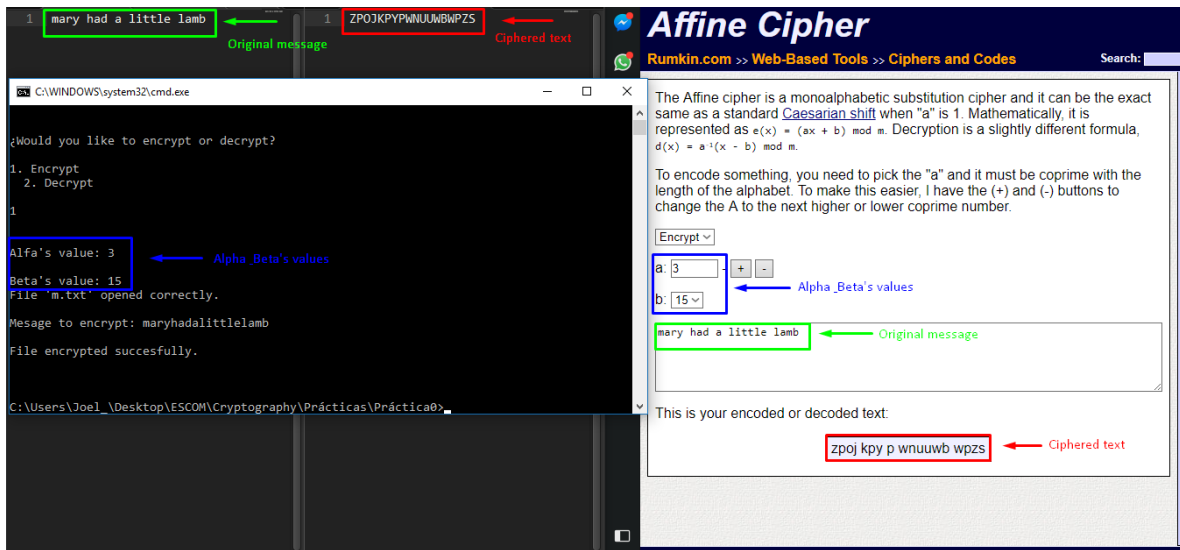


Figure 5. Comparison between a web page and my own program with the same values and messages.

Now, we proceed to verify the decryption option, first with wrong values and then with the correct ones to see if our decryption function described on Literature Review is correctly implemented.

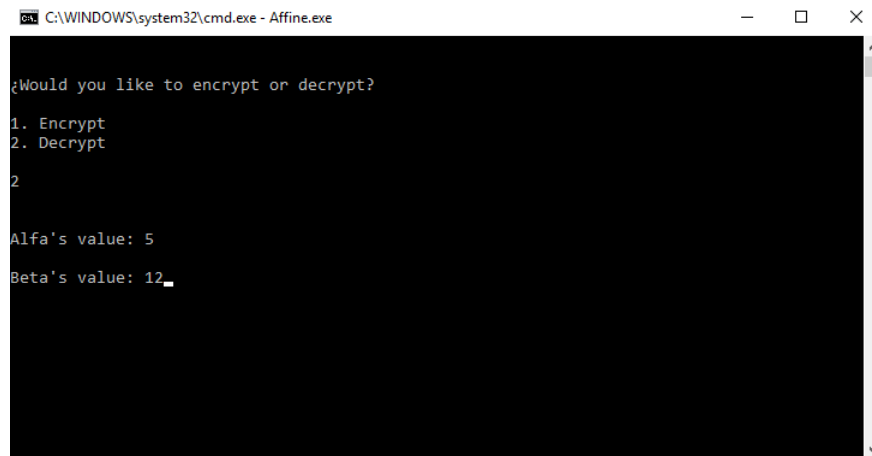


Figure 6. Main menu of the program selecting option 2 (Decrypt).

We can observe, that the value of alpha and beta are wrong (Figure 6), so, what must happen is that the message deciphered is incorrect, because the function won't be the correct one to decrypt the message correctly.

In the following image (Figure 7), we can see the “decrypted” message, but it’s unreadable because it has no sense, and the message we were expecting is “mary had a little lamb”, like in the original message written in the text file.

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

Would you like to encrypt or decrypt?
1. Encrypt
2. Decrypt
2

Alfa's value: 5
Beta's value: 12
File 'c.txt' opened correctly.
Message to decrypt: ZPOJKPYPWNUUWBWPZS
Decrypted message: 'nlqpklsicvmmcdclnw'

C:\Users\Joel\Desktop\ESCOM\Cryptography\Prácticas\Práctica0>
```

Figure 7. Wrong decrypted message due to alpha's and beta's values.

As we said before, the decrypted message is incorrect and make no sense, now, we'll try with the correct combination of values and see if it's correct the algorithm implemented in the “Code” Section.

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

Would you like to encrypt or decrypt?
1. Encrypt
2. Decrypt
2

Alfa's value: 3
Beta's value: 15
File 'c.txt' opened correctly.
Message to decrypt: ZPOJKPYPWNUUWBWPZS
Decrypted message: 'maryhadalittlelamb'

C:\Users\Joel\Desktop\ESCOM\Cryptography\Prácticas\Práctica0>
```

Figure 8. Correct decrypted message with the correct combination of alpha's and beta's values.

In the Figure 8 we can see that the correct combination (**key**) of the values on alpha and beta, gives us a readable and a coherent message, that was the original we tried in the text file showed before (Figure 5).

As you can see, the results are satisfactory even if the values are wrong because the message is protected by the key and the algorithm I implemented on the program, in the next section I will explain some little “errors” in the execution time and what's the importance of this practice nowadays (as I already said, it's a classical cipher, too weak in these days).

Extended Euclidean algorithm⁵ is implemented in Figure 9, is used to know the multiplicative inverse of alpha, only if greatest common divisor is 1, because it means that alpha and the alphabet size are prime each other.

```

C:\WINDOWS\system32\cmd.exe
26 = (3 * 8) + 2
3 = (2 * 1) + 1
2 = (1 * 2) + 0
3 = 13(0) + 3      || 1 = 1(1) - 0(0)
13 = 3(4) + 1      || 1 = 1(1) - 4(0)
3 = 1(3) + 0        || 1 = 13(1) - 4(-3)

Multiplicative Inverse:
File 'c.txt' opened correctly.
Message to decrypt: TMHG5DFMNGIH

Decrypted message: 'cryptography'

C:\Users\Joel_\Downloads\Practica0>

```

Figure 9. Implementation of Extended Euclidean Algorithm to find the multiplicative inverse of alpha

Discussion:

It's clear that Affine Cipher could be a great cipher (could think that), because there are 25^2 possible combinations with alpha and beta, but still being a classical substitution cipher.

However, the importance of encrypting a message is hire the information from a person not allowed to read it, and this (Affine), accomplish the objective, or at least, did it some years ago.

The results obtained in the previous section was correct, because the program read a file with the message you want to encrypt and then ask you for alpha's and beta's values to encrypt your message and write it on another text file to decrypt it at any moment. One important application is protecting personnel information like address, salary, credit card, etc.

Conclusions:

I learned that making a cipher is not too easy (but this practice was kind of), and is important to protect information always, even if you dedicate yourself to something else like singing, dancing, teaching, etc. And one of the needed services is provided by modern cryptography with the new algorithms that have been developed since at least, 40 years ago. For example, you could send your address and/or your phone number by one social network, but what happen if there's a guy who's taking information to kidnap your child.

One problem here, is that the program fails when the text to encrypt exceeds ± 600 characters, and we know that sensitive information has a longer length than that, so is a problem I need to solve immediately to offer my users more security in their personnel information.

Generally, my program could be applied at any alphabet, with some little fixes, but for example I implemented the alphabet size dynamic, so, we could have only a part of it and calculate correctly the ciphered/deciphered text from 2 simple values.

Of course, this program is not perfect, but I tried to optimize the most I could, for example avoiding the pair numbers in the function to obtain the multiplicative inverse because there's time to the processor we're wasting, in memory too.

References:

- [1] Instituto Nacional de Tecnologías de la Comunicación, 'LA CRIPTOGRAFÍA DESDE LA ANTIGUA GRECIA HASTA LA MÁQUINA ENIGMA', 2010. [Online]. Available: http://www.egov.ufsc.br/portal/sites/default/files/la_criptografia_desde_la_antigua_grecia_hasta_la_maquina_enigma1.pdf. [Accessed: 26 – August- 2017].
- [2] Franco Martínez Edgardo Adrián, 'Complejidad de los Algoritmos', 2017, [Online]. Available: <http://www.eafranco.com/docencia/analisisdealgoritmos/files/ejercicios/02.pdf>. [Accessed: 18 – August – 2017].
- [3] Tutorialspoint, 'The C Standard Library', 2012, [Online], Available: https://www.tutorialspoint.com/c_standard_library/. [Accessed: 27 – August – 2017].
- [4] Akins Tyler, 'Affine Cipher', 2008, [Online]. Available: <http://rumkin.com/tools/cipher/affine.php>. [Accessed: 27 – August – 2017].
- [5] Andrés Esquivel, 'IMPLEMENTACIÓN DEL ALGORITMO DE EUCLIDES EXTENDIDO EN JAVA', 2015, [Online]. Available: <http://blog.andresed.me/2015/06/implementacion-de-euclides-extendido-en.html>. [Accessed: 1 – September – 2017].

Code

Affine.c

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

int main (int argc, char const *argv[])
{
    menu ();
    return 0;
}
```

Functions.h

```
//Encryption functions
void encrypt (int alpha, int beta);
char * readMessage ();
void writeCiphertext (char * ciphertext);

//Decryption functions
void decrypt (int alpha, int beta);
char * readCiphertext ();
int alg_euc_ext(int n1,int n2);

//Shared encryption/decryption functions
void menu ();
void validateNumbers (int alpha, int beta);
int gcd (int alpha, int alphabet);
int multiplicativeInverse (int alpha);
int inverse_aditive (int beta);
```

Functions.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define ALPHABET_SIZE 26
#include "functions.h"

int i, value;

//ENCRYPTION FUNCTIONS
void encrypt (int alpha, int beta)
{
    char * message, * ciphertext = (char *) malloc (sizeof (char));
    validateNumbers (alpha, beta);
    message = readMessage ();

    //Receiving the message to encrypt and save it in a
    dynamic array
    printf("\nMessage to encrypt: %s\n\n", message);
    //Print the message to know it is correct
    for (i = 0; i < strlen (message); i++)
    {
        value = message [i] - 97;
```

```

        value *= alpha;
        value += beta;
        value %= ALPHABET_SIZE;
        ciphertext[i] = value + 65;
    }
    ciphertext[i] = '\0';
    writeCiphertext (ciphertext);
}

char * readMessage ()
{
    FILE * message;
    char c, * msgToEncrypt = (char *) malloc (sizeof (char));
    i = 0;
    message = fopen ("m.txt", "r");
    if (message == NULL)
    {
        printf("Error while opening file: 'm.txt'.\n");
    }
    else
    {
        c = fgetc (message);
        while (c != EOF)
        {
            if (c != 32 && c != '\n')
            {
                if ((c >= 'a' && c <= 'z'))
                {
                    msgToEncrypt[i++] = c;
                }
                else
                {
                    printf("Error, the file to encrypt has to has small letters only.\n");
                    exit (0);
                }
            }
            c = fgetc (message);
        }
        msgToEncrypt[i] = '\0';
        fclose (message);
        return msgToEncrypt;
    }
}

void writeCiphertext (char * ciphertext)
{
    FILE * encryptedMessage;
    encryptedMessage = fopen ("c.txt", "w");
    if (encryptedMessage == NULL)
    {
        printf("Error while creating file: 'c.txt'\n");
    }
    fprintf(encryptedMessage, "%s", ciphertext);
    printf("File encrypted succesfully.\n\n\n");
}

```

```

        fclose (encryptedMessage);
        //We close the file after reading it
    }

//DECRYPTION FUNCTIONS
void decrypt (int alpha, int beta)
{
    char * ciphertext, * plaintext = (char *) malloc (sizeof (char));
    validateNumbers (alpha, beta);
    int inverse, additive_inverse;
    inverse = alg_euc_ext(alpha,beta);
    //Obtaining the multiplicative inverse for alpha
    additive_inverse = inverse_aditive (beta);
    ciphertext = readCiphertext ();
    //Receiving the message to decrypt and save it in a
    dinamic array
    printf("\nMessage to decrypt: %s\n\n", ciphertext);
    //Print the message to know it is correct
    for (i = 0; i < strlen (ciphertext); i ++)
    {
        value = ciphertext [i] - 65;
        value *= inverse;
        //Multiplying each letter by multiplicative
        inverse of alpha
        value += (additive_inverse * inverse);
        //Adding the aditive inverse of beta
        value %= ALPHABET_SIZE;
        //We get the value module alphabet's size
        plaintext [i] = value + 97;
        //We save each decrypted letter in a dinamic array
    }
    plaintext [i] = '\0';
    //We add null character to avoid trash on
    the array
    printf("\nDecrypted message: '%s'\n\n\n", plaintext);
    //Finally, we show the original message to the user
}

int inverse_aditive (int beta)
{
    for (i = 1; i < ALPHABET_SIZE; i ++)
    {
        if ((i + beta) % ALPHABET_SIZE == 0)
            return i;
    }
}

int multiplicativeInverse (int alpha)
{
    int x, inverse;
    for(inverse = 0; inverse < ALPHABET_SIZE; inverse++)
    {
        x = (alpha * inverse) % ALPHABET_SIZE;
        if(x == 1)
            return inverse;
    }
}

char * readCiphertext ()
{
    FILE * message;
    //Pointer for read the file
    char c, * msgToDecrypt = (char *) malloc (sizeof (char));
    i = 0;
    message = fopen ("c.txt", "r");
    //Opening the file in reading mode
    if (message == NULL)
        printf("Error while opening file: 'c.txt'.\n");
    else
        printf("File 'c.txt' opened correctly.\n");
}

```

```

        c = fgetc (message);
        //Reading the first character
        while (c != EOF)
            //While it's not the end of the file
            {
                if (c != 32 && c != '\n')
                    //If is a space or a line break, we don't add it
                    {
                        if ((c >= 'A' && c <= 'Z'))
                            msgToDecrypt [i ++] = c;
                        //We save capital letters on our dinamic array
                        else
                            {
                                //If it's
                                another character, we end the program
                                printf("Error, the file to decrypt has been
                                modified.\n");
                                exit (0);
                            }
                        c = fgetc (message);
                        //Reading the next character
                    }
                msgToDecrypt [i] = '\0';
                //We add null character to avoid trash on the array
                fclose (message);
                //We close the file after reading it
                return msgToDecrypt;
                //Return the encrypted message to decrypt
            }
    }
    from file
}

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 aux;
    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    ||
                1 = %d(%d) - %d(%d)  \n",n1*q+r,n1,q,r,x1,y2,x2,y1 );
            }else{
                printf("%d = %d(%d) + %d    ||
                1 = %d(%d) - %d(%d)  \n",n1*q+r,n1,q,r,x2,y1,x1,y2 );
            }
            flag++;
        }

        array[0] = n1;
        array[1] = x2;
        array[2] = y2;
    }

    aux = multiplicativeInverse(in);

    return aux;
}

```

```

}

//SHARED ENCRYPTION/DECRYPTION FUNCTIONS
void menu ()
{
    int option, alpha, beta;
    system ("cls");
    printf("\n\n%cWould you like to encrypt or decrypt?\n\n", 168);
    printf("1. Encrypt\n2. Decrypt\n\n");
    scanf ("%d", &option);
    printf("\n\nAlfa's value: ");
    scanf ("%d", &alpha);
    printf("\n\nBeta's value: ");
    scanf ("%d", &beta);
    system ("cls");
    if (option == 1)
        encrypt (alpha, beta);
    //If option 1, we encrypt the message
    else if (option == 2)
        decrypt (alpha, beta);
    //If option 2, we decrypt the message
    else
        menu ();
}

void validateNumbers (int alpha, int beta)
{
    if (alpha == 1)
    {
        if (beta == ALPHABET_SIZE)
        {
            printf("Error, the text won't encrypt/decrypt correctly due to
alpha's and beta's values.\n\nAlfa: %d \t Beta: %d.\n\n", alpha, beta);
            menu ();
        }
        else if (alpha <= 0)
            //If alpha's value is less than zero, we
return to the menu
        {
            printf("Error, alpha's value must be between 1 and alphabet's size.\n");
            menu ();
        }
        if (beta <= 0 || beta > ALPHABET_SIZE)
            //If beta's value is bigger than alphabet size, we return to the menu
        {
            printf("Error, beta's value must be between 1 and alphabet's size.\n");
            menu ();
        }
        if (gcd (alpha, ALPHABET_SIZE) != 1)
            //We calculate the greatest common divisor
        {
            printf("Error, the text won't encrypt/decrypt correctly due to alpha's
value.\n");
            menu ();
        }
    }
}

int gcd (int alpha, int alphabet)
    //Implementation of Euclides algorithm to obtain the greatest
common divisor
{
    int temp, inverse, x;
    if (alpha > alphabet)
    {
        temp = alphabet;
        alphabet = alpha;
        alpha = temp;
    }
    if (alpha != 0)
        //While alpha's value is bigger than zero
    {

```

```
        printf ("%d = (%d * %d) + %d\n", alphabet, alpha, alphabet / alpha, alphabet -  
(alphabet/alpha) * alpha);  
        gcd (alpha, alphabet % alpha);  
    }else  
    {  
        return alphabet;  
    }  
}
```