Grupo B

Barbero Angulo, Vicente	15502793B
García-Tenorio Fernández, Sergio	05455863X
Esteban Gracia, Javier	26280026L
Monje Lola, Oscar Angel	49100617-X
de Lanzas Garcia, Carlos	21003452- J

CRIPTOGRAFÍA

Informe sobre la implementación de algoritmos de cifrado/descifrado básicos en FORTRAN90, cubriendo la descripción, validación del código y ejemplos de uso.

El código puede ser encontrado en
https://github.com/javiyo/CODIFICADOR

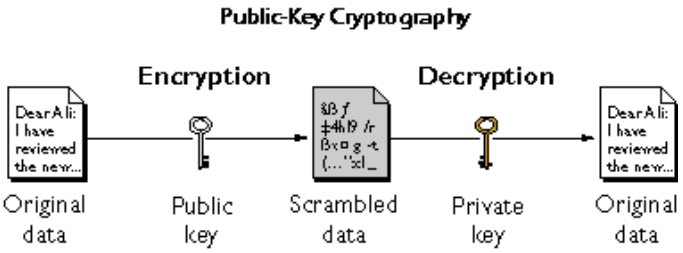
Descargar desde el terminal:

git clone https://github.com/javiyo/CODIFICADOR

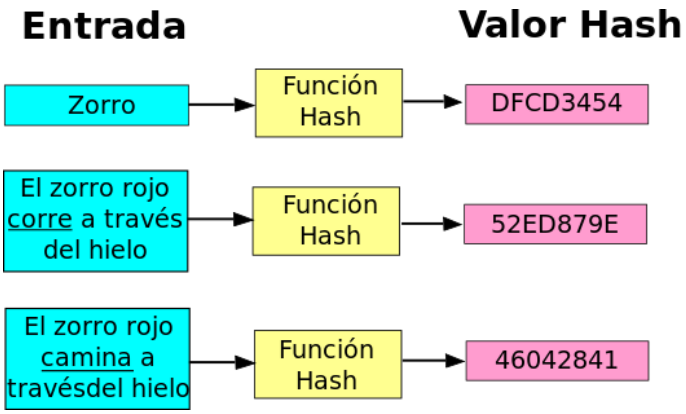
Introducción

En este programa se van a aplicar distintas técnicas de cifrado, que aunque en desuso actualmente por lo simple que es desencriptarlas, van a sernos útiles para entender un poco alguno de los principios de la criptografía.
Se usaran tres técnicas cada una algo más complicada; el cifrado aditivo, el multiplicativo y afín, explicaciones más detalladas del funcionamiento de estas se cubrirán a continuación, actualmente no se usan, ya que es posible con relativa facilidad encontrar el mensaje sin tener la clave ya que incluso con el cifrado más avanzado de estos, solo tenemos 392 claves distintas, algo perfectamente descifrable por una persona humana en bastante menos tiempo de su esperanza de vida.

Actualmente lo más común son sistemas de cifrado basados en la dificultad computacional que supone factorizar números primos muy grandes, con lo que se llama criptografía de clave pública (Wikipedia, s.f.)



Cuando se trata de guardar claves en una base de datos, por ejemplo se suelen utilizar otro tipo de algoritmos como SHA1, o MD5 (actualmente poco seguros) que lo que son es una función que genera un código a partir de la información que le metes, pero a diferencia del que estamos viendo es muy difícil obtener la clave inversa, por lo tanto la única forma de saber si una palabra es el mensaje cifrado es sabiendo la palabra de antemano y probando si el hash coincide. Para sacar estas contraseñas se suelen usar diccionarios con miles de palabras, o bien por fuerza bruta lo que requiere gran capacidad de computación. (Wikipedia, s.f.)



Descripción del código

Cifrador

El programa “cifrador” utiliza el modulo criptografía. El programa pide que selecciones una de las 3 técnicas que puede emplear. A continuación se declara inicialmente que i=0 y se crea un bucle que funcione mientras se cumpla esta afirmación. Si la técnica es “3”, se piden dos claves de cifrado, en caso contrario se pide solo una. En el final del programa hemos puesto un un *else* de modo que si la técnica no es “1”, “2” o “3” muestre un mensaje de error. En cuanto a las claves en ambas cogemos el valor absoluto del número introducido y en caso de que sean divisibles entre “2” o “13” el programa muestre otro mensaje diciendo que no son válidas.

```
i=0
do while(i==0)
        if (tecnic==3) then
                print *, 'Introduce las claves de cifrado'
                read *, k,b
        else            !Para aditivo y multiplicativo solo es necesario una clav
                print *, 'Introduce la clave de cifrado'
                read *, k
        end if
        k=abs(k)
        b=abs(b)
        if ((mod(k,2)==0.OR.mod(k,13)==0).AND.(tecnic==3.OR.tecnic==2)) then !Los
                print *, 'Error, con esa clave es imposible encontrar inversa'
        else
                i=1
        end if
end do
```

La siguiente parte se encarga de la lectura de los nombres de los archivos, y la correspondiente concatenación de extensión para abrir. Además guarda como de extenso es el texto.

```
print *, '¿Como se llama el archivo con el mensaje que quieres descifrar?(sin extension)' !Introduccion de
read *, archivociph
print *, '¿Como quieres llamar al archivo con el mensaje descifrado?(sin extension)'
read *, archivoplano
                    !Apertura de los archivos cifrados y del archivo donde se va a descrifrar, la extensior
                open(unit=11, file=trim(archivociph)//".cfr",status="old")
                open(unit=12, file=trim(archivoplano)//".dcf")

long=larguitud(11) !En este momento fijamos la longitudo de nuestro mensaje gracias a la funcion larguitud
```

Por último el programa ve que técnica hemos escogido, y va leyendo las letras una a una y escribiendo la transformada de la letra una a una en el archivo de destino

```
if (tecnic==1) then
        do i=1,long
        read(11, "(A1)",advance='no')letra
        write(12, "(A1)",advance='no')numletra(aditivo(letranum(letra),k)) !Va llamando
        end do
else if (tecnic==2) then
        do i=1,long
        read(11, "(A1)",advance='no')letra
        write(12, "(A1)",advance='no')numletra(multiplicativo(letranum(letra),k))
        end do
else if (tecnic==3) then
        do i=1,long
        read(11, "(A1)",advance='no')letra
        write(12, "(A1)",advance='no')numletra(afin(letranum(letra),k,b))
        end do
```

Descifrador

Esencialmente el mismo programa con la misma estructura con la única diferencia que los archivos cambian de nombre y de extensión y en vez de las funciones con la clave deberemos aplicarles la clave inversa:

```
                open(unit=11, file=trim(archivociph)//".cfr",status="old")
                open(unit=12, file=trim(archivoplano)//".dcf")

long=larguitud(11) !En este momento fijamos la longitudo de nuestro mensaje gracias a la funcion larguitud

if (tecnic==1) then !Proceso de lectura del archivo cifrado y posterior escritura del archivo descifrado letr
        do i=1,long
        read(11, "(A1)",advance='no')letra
        write(12, "(A1)",advance='no')numletra(aditivo(letranum(letra),rkaditivo(k))) !Va llamando a las dist
        end do
else if (tecnic==2) then
        do i=1,long
        read(11, "(A1)",advance='no')letra
        write(12, "(A1)",advance='no')numletra(multiplicativo(letranum(letra),rkmultiplicativo(k)))
        end do
else if (tecnic==3) then
        call rkafin(k,b)
        do i=1,long
        read(11, "(A1)",advance='no')letra
        write(12, "(A1)",advance='no')numletra(afin(letranum(letra),k,b))
        end do
else
        print *, "Error, seleccion de tecnica incorrecta" !Error que se da cuando se introduce un numero dist
end if
```

Modulo

El modulo es la base de nuestro programa, el cual contiene las funciones que hacen funcionar perfectamente a nuestro programa:

-Letranum: En esta función tenemos definido un vector que contiene las 26 letras del abecedario. Esta, recibe una variable, una letra, a la que le imponemos una condición. La condición es que si esta letra recibida coincide con una de las letras que contiene el vector, entonces, la variable "letranum" tomara el valor de la posición en la que se encuentra la letra en el vector.

```
function letranum(a) !Funcion que asigna a cada letra un número para su posterior cifrado
character, intent(in) :: a
integer :: letranum, i
character,dimension(26) :: v
v=(/"a","b","c","d","e","f","g","h","i","j","k","l","m","n","o","p","q","r","s","t","u","v","w","x","y","z"/)
do i=1,26
        if (a==v(i)) letranum=i
end do
```

-Numletra: Esta función desempeña la tarea de convertir una variable numérica que recibe (aditivo/multiplicativo/afín), en una letra. La variable numérica obtenida, representa la posición que ocupa en el vector la letra del mensaje ya cifrada.

```
function numletra(n) !Función que tras el cifrado,asigna el número obtenido con la técnica de cifrado a la nueva letra.
integer, intent(in) :: n
character :: numletra
character,dimension(26) :: v
v=(/"a","b","c","d","e","f","g","h","i","j","k","l","m","n","o","p","q","r","s","t","u","v","w","x","y","z"/)
numletra=v(n)
end function
```

-Aditivo: Tiene dos variables de entrada, el número de la función "letranum" y la clave de cifrado. Funciona de la siguiente manera: Se suman la clave a el código de la letra y se resta 26 hasta que el número este entre 1 y 26

```
function aditivo(a,k) !Función de cifrado aditivo
integer, intent(in) :: a, k
integer :: ad, aditivo  !El aditivo suma la clave a nuestro numero
ad = a + k
do while (ad>26) !Si tras la adicion,obtenemos un número mayor que 26,rest
ad= ad-26
end do
aditivo = ad
end function
```

-Multiplicativo: Esta función tiene, como la anterior, dos variables de entrada, la función "letranum" y la clave de cifrado. Las cuales se multiplican y resultan en la variable "mul". El resto entre "mul" y 26, será la posición en el vector de la letra ya cifrada. En afin y en multiplicativo el resto puede dar 0 (esto en realizad es para nosotros la z)

```
function multiplicativo(a,k) !Simple func
integer, intent(in) :: a,k
integer :: mul, multiplicativo
mul = k*a
multiplicativo=mod(mul,26)      !Como el
if (multiplicativo==0) multiplicativo=26
end function multiplicativo
```

-Afín: Tiene 3 variables de entrada. Las claves de cifrado de multiplicativo y aditivo y la función letranum. Esta función multiplica la clave de cifrado de multiplicativo por lo que valga la función letranum y después, le suma la clave de cifrado aditivo. Esta función es la más compleja ya que combina las dos técnicas de encriptación anteriores.

```
function afin(a,k,b)
integer, intent(in) :: a,k,b
integer :: afin                 !Afin (realiza el multiplicativo y le sum
afin=mod((k*a)+b,26)
if (afin==0) afin=26 !Cuando k*a+b sea igual a 26 el resto es 0, por lo c
end function afin    !para asignarlo a la z, tendremos que cambiar el 0 p
```

-Rkaditivo: Es la función inversa a la aditivo, la utilizada en el decodificador. Tiene un variable de entrada, la clave de "cifrado", la clave inversa se obtendrá reduciendo la de entrada y restándole 26.

```
function rkaditivo(k) !Funcion que calcula la clave inversa en el cifrado aditivo
integer,intent(in) :: k
integer :: rkaditivo, ad
ad=k
do while (ad>26) !Restamos 26,ya que nuestra tabla con el abecedario solo tiene 26 caracteres
ad=ad-26
end do
rkaditivo=26-ad
end function rkaditivo
```

-Rkmultiplicativo: Esta es una función compleja. Tiene una variable de entrada, la clave de cifrado de multiplicativo. Con la condición que k*rk=1mod26 (Schwartz) podemos ver que tiene que haber un número para que el número cuya división entre 26 tenga resto 1 es divisible por la clave.

```
function rkmultiplicativo(k) !Función que calcula la clave inversa en el cifrado multiplicativo
integer, intent(in) :: k
integer :: rkmultiplicativo, i=1
do while (mod(26*i+1,k)/=0)!Este bucle hace satisfacer la condicion necesaria de que k*rk=1mod26 (siendo
i=i+1
end do
rkmultiplicativo=(26*i+1)/k
end function rkmultiplicativo
```

-Rkafin: Contamos con una subrutina que lo que hace es convertir ambas claves de cifrado introducidas en el decodificador(aditivo y multiplicativo), en sus claves inversas de cifrado con las dos funciones anteriores. De nuevo usando el método visto en (Schwartz)

```
subroutine rkafin(k,b) !Función que calcula las claves inversas en el cifrado afín, combinacion de las funciones anteriores
integer :: k, b
k=rkmultiplicativo(k) !Aplica el algoritmo para encontrar la clave inversa
b=rkaditivo(b)*k
do while (b>26)
b= b-26
end do
end subroutine
```

-Larguitud: Esta función cuenta el número de caracteres que hay en el mensaje a cifrar o descifrar. Utiliza la función "len_trim", que cuenta los caracteres sin espacios para añadir dificultad al descifrado del mensaje. Además, se impone la condición de que el texto no debe tener más de 80 caracteres.

```
function larguitud(a)!Función que cuenta el numero de carácteres a cifrar
integer, intent(in) :: a
character(len=32767) :: texto
read(a,*)texto
rewind(a)
larguitud=len_trim(texto)!Cuenta los caracteres del texto sin espacios
if (larguitud>80) then
        print*,"ERROR; El texto a cifrar debe tener menos de 80 caracteres"
        stop
end if
end function larguitud
```

Validación del código desarrollado

Para validar el código se han probado multitud de casos, tanto para claves extremas y siempre codificando el abecedario para asegurarnos de que no haya ninguna letra que falle en el cifrado/descifrado.

Una forma fácil de comprobar que el programa lo hace correctamente es introducir un texto del que ya sabemos lo que tiene que cifrar para validar el cifrador. Nos ayudaremos del cipher manual y probamos un texto, el resultado era el mismo.

```
if (tecnic==1) then
        do i=1,80
        read(11, "(A1)",advance='no')letra
        write(12, "(I2,x)",advance='no')aditivo(letranum(letra),k)
        end do
else if (tecnic==2) then
        do i=1,80
        read(11, "(A1)",advance='no')letra
        write(12, "(I2,x)",advance='no')multiplicativo(letranum(letra),k)
        end do
else if (tecnic==3) then
        do i=1,80
        read(11, "(A1)",advance='no')letra
        write(12, "(I2,x)",advance='no')afin(letranum(letra),k,b)
        end do
end if
```

Este programa nos valida que las claves cifradas son las correctas con cada una de las técnicas ya que nos muestra el código numérico de cada letra que codificamos, para solucionar posibles fallos, en el programa de descifrado una vez visto que es correcto el cifrado podemos descifrar un mensaje cifrado y ver que funciona bien.

En cuanto a validar las funciones y subrutinas es tan sencillo como darle valores conocidos y comprobar que funcionan correctamente con casos diversos.

También se ha comprobado que a la hora de poner claves en la parte multiplicativa y afín los multiplos de “13” y los números pares nos sacan del programa y nos piden volver a poner la clave, además de que si el texto tiene más de 80 caracteres nos mostrará un mensaje de error y el programa se parará sin cifrar/descifrar nada.

```
¿Que tecnica de cifrado quieres emplear?
 Para aditivo introduzca 1
 Para multiplicativo introduzca 2
 Para afin introduzca 3
3
 Introduce las claves de cifrado
39
11
 Error, con esa clave es imposible encontrar inversa
 Introduce las claves de cifrado
```

```
 ¿Que tecnica de cifrado quieres emplear?
 Para aditivo introduzca 1
 Para multiplicativo introduzca 2
 Para afin introduzca 3
2
 Introduce la clave de cifrado
31
 ¿Como se llama el archivo con el mensaje a cifrar?(sin extension)
plaintext
 ¿Como quieres llamar al archivo con el mensaje cifrado?(sin extension)
cifra
 ERROR; El texto a cifrar debe tener menos de 80 caracteres
```

Esto nos ha ayudado a solucionar errores del programa que evitaban que ciertas letras mostrasen otro resultado distinto por pantalla.

Ejemplos de uso

Un uso real de este programa consistiría en la comunicación entre dos personas de forma que el "canal" es abierto, pero podemos conseguir que para los desconocedores de las claves usadas parezca un sinsentido, es por esto que es esencial que los dos seres que se están comunicando hayan acordado previamente las claves que se van a utilizar, ya sea directamente o tomando alguna referencia con algún método que permita saber qué clave se va a usar en algún momento ya sea acordando en el tiempo y cambiando las claves cada cierto tiempo a otras pre acordadas para mejorar la poca seguridad que nos dan estos métodos.

I)

Técnica usada: cifrado aditivo con clave 5.

Texto original: 'beatihispaniquibusvivereestbibere'
Texto cifrado:'gjfynmnxufsnvzngzxanajwjjxygngjwj'
Texto descifrado:'beatihispaniquibusvivereestbibere'

II)

Técnica usada: cifrado multiplicativo con clave 5

Texto original: 'dequecoloreselcaballoblancodesantiago'
Texto cifrado: 'tygayowhwlyqyhoejehhwjherowtyqervseiw'
Texto descifrado: 'dequecoloreszelcaballoblancodesantiago'

III)

Técnica usada: cifrado afín con claves 321 y 123 (caso extremo)
Texto original: 'lamenteescomounparacaidassolofuncionasiseabre'
Texto cifrado: 'wbfloqllhtxfxzogbybtbvcbhhxwxuzotvxobhvhlbkyl'
Texto descifrado: 'lamenteescomounparacaidassolofuncionasiseabre'

Conclusión

Descifrar mensajes que se han codificado con estas técnicas es muy sencillo, saber la frecuencia de aparición de letras en el abecedario prácticamente nos deja con muy poquitas posibilidades, pero aun así es mucho mas fácil si realizamos un programa que nos imprima todos los mensajes con las combinaciones de respuestas posibles, el afin con clave “k” 1 y “b” [1,26] sería el equivalente al aditivo y cuando el aditivo suma 26, sería equivalente a tener únicamente un multiplicativo, además de todas las demás posibilidades que se encuentran en el afin.

```
fiexmlmwtermuymfywzmziviiwxfmfivi _/          1 /          1
ehdwlklvsdqltxlexvylyhuhhvwelehuh _/          1 /          2
dgcvkjkurcpkswkdwuxkxgtgguvdkdgtg _/          1 /          3
cfbujijtqbojrvjcvtwjwfsfftucjcfsf _/          1 /          4
beatihispaniquibusvivereestbibere _/          1 /          5
adzshghrozmhpthatruhudqdddrsahadqd _/         1 /          6
zcyrgfgqnylgosgzsqtgtcpccqrzgzcpc _/          1 /          7
ybxqfefpmxkfnrfyrpsfsbobbpqyfybob _/          1 /          8
xawpedeolwjemqexqoreranaaopxexana _/          1 /          9
wzvodcdnkvidlpdwpnqdqzmzznowdwzmz _/          1 /          10
vyuncbcmjuhckocvompcpylyymnvcvyly _/          1 /          11
uxtmbablitgbjnbunloboxkxxlmubuxkx _/          1 /          12
twslazakhsfaimatmknanwjwwkltatwjw _/          1 /          13
svrkzyzjgrezhlzsljmzmvivvjkszsviv _/          1 /          14
ruqjyxyifqdygkyrkilyluhuuijryruhu _/          1 /          15
qtpixwxhepcxfjxqjhkxktgtthiqxqtgt _/          1 /          16
psohwvwgdobweiwpigjwjsfssghpwpsfs _/          1 /          17
orngvuvfcnavdhvohfivirerrfgovorer _/          1 /          18
nqmfutuebmzucgungehuhqdqqefnunqdq _/          1 /          19
mpletstdalytbftmfdgtgpcppdemtmpcp _/          1 /          20
lokdsrsczkxsaeslecfsfoboocdlslobo _/          1 /          21
knjcrqrbyjwrzdrkdberenannbckrknan _/          1 /          22
jmibqpqaxivqycqjcadqdmzmmabjqjmzm _/          1 /          23
ilhapopzwhupxbpibzcpclyllzaipilyl _/          1 /          24
hkgzonoyvgtowaohaybobkxkkyzhohkxk _/          1 /          25
gjfynmnxufsnvzngzxanajwjjxygngjwj _/          1 /          26
bcshmdmyxsfmgqmbqyzmzcpccyhbmbcpc _/          3 /          1
stjydudpojwdxhdshpqdqtgttpysdstgt _/          3 /          2
jkapulugfanuoyujyghuhkxkkgpjujkxk _/          3 /          3
abrglclxwrelfplapxylybobbxgalabob _/          3 /          4
rsixctconivcwgcrgopcpsfssoxrcrsfs _/          3 /          5
ijzotktfezmtnxtixfgtgjwjjfoitijwj _/          3 /          6
zaqfkbkwvqdkeokzowxkxanaawfzkzana _/          3 /          7
qrhwbsbnmhubvfbqfnoborerrnwqbqrer _/          3 /          8
hiynsjsedylsmwshwefsfiviienhshivi _/          3 /          9
yzpejajvupcjdnjynvwjwzmzzveyjyzmz _/          3 /          10
pqgvaramlgtaueapemnanqdqqmvpapqdq _/          3 /          11
```

```
k=1
long=larguitud(11)
do while (k<26)
        if (k/=13) then
        do b=1,26
                r=k
                s=b
                call rkafin(r,s)
                do i=1,long
                read(11, "(A1)",advance='no')letra
                write(12, "(A1)",advance='no')numletra(afin(letranum(letra),r,s))
                end do
                rewind(11)
                write(12,*)"_/",k,"/",b
        end do
        end if
        k=k+2
end do
```

Todos los métodos de cifrado citados anteriormente son, en general, poco seguros. Existen, sin embargo, métodos que son matemáticamente indescifrables, como es el caso de la libreta de un solo uso. Inventada en 1917 y si es usada correctamente es indescifrable, tanto que se sigue usando, y incluso puedes seguir escuchando “estaciones de números” en radio HF. (Wikipedia)

Referencias

Wikipedia. (s.f.). Obtenido de https://en.wikipedia.org/wiki/One-time_pad

Schwartz, •. S. (s.f.). *Criptology for Beginners.* Obtenido de http://www.mastermathmentor.com/mmm/Crypt.ashx

Wikipedia. (s.f.). Obtenido de https://en.wikipedia.org/wiki/Public-key_cryptography

Wikipedia. (s.f.). Obtenido de https://es.wikipedia.org/wiki/Funci%C3%B3n_hash

Tutorialspoint.com. Obtenido de https://www.tutorialspoint.com/fortran/fortran_characters.htm