INSTITUTO POLITÉCNICO NACIONAL
CENTRO DE INVESTIGACION EN COMPUTACION

Introducción a la Criptografía.

Cifrado de Bloque

PROFESOR: Gallegos García Gina

Alumna: Cruz Chavez Alan Francisco

Contenido

Introducción

En el presente documento se anexarán las capturas de pantalla además de la explicación del código que resuelve un ejercicio del Cifrado de Bloque desarrollado en el lenguaje de programación Python, así como las capturas donde se va observando cómo es que el algoritmo se va desarrollando.

Desarrollo

Planteamiento del Problema

El problema que se resolverá con el Cifrado de Bloque será el siguiente el cual fue proporcionado por la Dra. Gina:

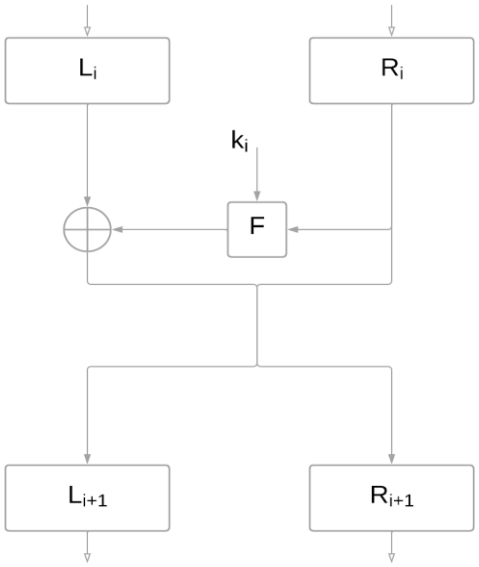
El algoritmo usara bloques de tamaño de 8 caracteres. Tendrá dos vueltas y en cada vuelta realizará una operación de sustitución S y una de permutación P sobre la 1ª mitad.

Sustitución: $C_i = (M_i + 1) mod\ 27$.

Permutación: $C_j = \Pi_{3241}$ *(el carácter 1° pasa a la 4° posición en el criptograma, el 4° a la 3°, el 2° a la 2° y el 3° a la 1°).*

Mensaje: M = CRIPTOGRAFIA ES UNA CIENCIA

A continuación, en la *Ilustración 1* se muestra el diagrama a bloque que se usa en el Cifrado de Bloque.



Donde:

$$L_{i+1} = R_i$$

$$R_{i+1} = L_i \oplus F(R_i, k_i)$$

Ilustración 1 - Diagrama a bloques del algoritmo.

Definición de datos

Empezando con la elaboración del programa, el primer paso es definir todas las variables globales que usaremos en el desarrollo de la implementación del algoritmo, el lenguaje en el que se desplego el algoritmo fue el de Python por su versatilidad y principalmente por el tipo de estructura de datos llamado *Diccionario*[1].

En la *Ilustración 2* se representa el Diccionario que nos dirá el valor de un carácter en base 10.

```
# abe_num es un diccionario que nos regrese el valor en entero dependiendo de la letra del abecedario
abe_num = {
    'A':  0, 'B':  1, 'C':  2, 'D':  3,
    'E':  4, 'F':  5, 'G':  6, 'H':  7,
    'I':  8, 'J':  9, 'K': 10, 'L': 11,
    'M': 12, 'N': 13, 'Ñ': 14, 'O': 15,
    'P': 16, 'Q': 17, 'R': 18, 'S': 19,
    'T': 20, 'U': 21, 'V': 22, 'W': 23,
    'X': 24, 'Y': 25, 'Z': 26
}
```

Ilustración 3 - Diccionario del Abecedario a Numero.

En la *Ilustración 3* se representa el Diccionario que nos da la equivalencia de un número a un arreglo con la equivalencia en binario.

```
# num_binario es un diccionario que nos regresa la equivalencia en binario de un numero
num_bin = {
    0:  '00001', 1:  '00010', 2:  '00011', 3:  '00100',
    4:  '00101', 5:  '00110', 6:  '00111', 7:  '01000',
    8:  '01001', 9:  '01010', 10: '01011', 11: '01100',
    12: '01101', 13: '01110', 14: '01111', 15: '10000',
    16: '10001', 17: '10010', 18: '10011', 19: '10100',
    20: '10101', 21: '10110', 22: '10111', 23: '11000',
    24: '11001', 25: '11010', 26: '11011'
}
```

Ilustración 3 - Diccionario de Numero a Binario.

[1] Un diccionario Python es una estructura de datos para trabajar con colecciones de datos almacenados en pares de claves/valores.

Finalmente, en la *Ilustración 4* se representa el Diccionario que nos da la equivalencia de un arreglo binario y nos devuelve una letra del abecedario.

```
# bin_abe es un diccionario que nos regrese una letra dependiendp de un arreglo binario
bin_abe = {
    '00001': 'A', '00010': 'B', '00011': 'C', '00100': 'D',
    '00101': 'E', '00110': 'F', '00111': 'G', '01000': 'H',
    '01001': 'I', '01010': 'J', '01011': 'K', '01100': 'L',
    '01101': 'M', '01110': 'N', '01111': 'Ñ', '10000': 'O',
    '10001': 'P', '10010': 'Q', '10011': 'R', '10100': 'S',
    '10101': 'T', '10110': 'U', '10111': 'V', '11000': 'W',
    '11001': 'X', '11010': 'Y', '11011': 'Z'
}
```

Ilustración 4 - Diccionario de Arreglo binario a Carácter.

Definición de Funciones

Empezamos definiendo los métodos que se usan en el algoritmo, el primero es el de Sustitución o mejor conocido en la materia como Confusión el cual se muestra en la *Ilustración 5.*

En la cual tenemos como argumentos de entrada los siguientes:

- arr4car: Es un arreglo de caracteres (la porción del bloque correspondiente a L_i).
- dicAbeNum: Es el diccionario abe_num definido en la sección de "Definición de datos".
- dicNumBin: Es el diccionario num_bin definido en la sección de "Definición de datos".

Y como argumentos de salida tenemos:

- listAux: La cual nos devuelve una *Lista*[2] que contiene string en Binario de la equivalencia de un carácter del Abecedario a su equivalencia en Binario.

```
def Sustitucion(arr4Car: str, dicAbeNum: dict, dicNumBin: dict) -> list:
    lisAux = []

    for elemnt in arr4Car:
        valNum = dicAbeNum[elemnt]
        valNumSust = (valNum + 1) % 27
        valBin = dicNumBin[valNumSust]
        lisAux.append(valBin)

    return lisAux
```

Ilustración 5 - Función Sustitución: Que es la equivalencia de la siguiente función $C_i = (M_i + 1) mod\ 27$.

En la *Ilustración 6* se muestra la función de Permutación la cual tiene los siguientes argumentos:

Argumentos de entrada:

- Lista: La lista que se obtiene de la función anterior.

Argumentos de salida:

- listAux: Es una lista en la cual se per mutan el orden en la que la lista de entrada llega.

```
def Permutacion(lista: list) -> list:
    listaAux = []

    listaAux.append(lista[2])
    listaAux.append(lista[1])
    listaAux.append(lista[3])
    listaAux.append(lista[0])

    return listaAux
```

Ilustración 6 – Función Permutación: La cual es la equivalencia de la siguiente función $C_j = \Pi_{3241}$.

En la *Ilustración 7* definimos la Función ConvertirAbeBin la cual usamos para pasar un arreglo de caracteres a una lista con arreglos Binarios, en esta función se definen los siguientes argumentos de entrada y salida.

Argumentos de entrada:

- arreglo: Es un arreglo de caracteres (la porción del bloque correspondiente a R_i).
- dicAbeNum: Es el diccionario abe_num definido en la sección de "Definición de datos".
- dicNumBin: Es el diccionario num_bin definido en la sección de "Definición de datos".

Y como argumentos de salida tenemos:

- listAux: La cual nos devuelve una *Lista*[2] que contiene string en Binario de la equivalencia de un carácter del Abecedario a su equivalencia en Binario.

[2] Listas son similares a matrices (o arrays) que se encuentran en otros lenguajes. Sin embargo, en Python se manejan como variables con muchos elementos.

```python
def ConvertirAbeBin(arreglo: str, dicAbeNum: dict, dicNumBin: dict)-> list:
    listAux = []

    for caracter in arreglo:
        valNum = dicAbeNum[caracter]
        valBin = dicNumBin[valNum]
        listAux.append(valBin)

    return listAux
```

Ilustración 7 – Función ConvertirAbeBin.

En la *Ilustración 8* se define la función AplicarMod27 la cual definimos ya que en la operación de la XOR (OR exclusiva) puede que el arreglo binario que nos resulte sea mayor a el numero 27 por lo cual debemos hacer la operación para saber a qué numero en base 10 pertenece dicho arreglo y poder aplicarle el módulo de 27 para que quede dentro de nuestro Diccionario, en esta función se definen los siguientes argumentos de entrada y salida.

Argumentos de entrada:

- strBin: Es un arreglo de caracteres binario.
- dicNumBin: que es el Diccionario num_bin.

Argumentos de salida:

- str: El arreglo de caracteres binario después de aplicar el módulo de 27 al arreglo de entrada.

```python
def AplicarMod27(strBin: str, dicNumBin: dict) -> str:
    numero_decimal = 0

    for posicion, digito_sting in enumerate(strBin[::-1]):
        numero_decimal += int(digito_sting) * 2 ** posicion

    valor = numero_decimal % 27

    return dicNumBin[valor]
```

Ilustración 8 – Función AplicarMod27.

En la *ilustración 9* definimos la última función de conversión nombrada ConvertirBinAbe la cual nos devuelve la equivalencia de un numero binario a un carácter del Abecedario, sus argumentos de entrada y salida son los siguientes.

Argumentos de entrada:

- listArr: Es una lista de arreglos de caracteres binario.
- dicBinAbe: Es el Diccionario bin_abe

Argumentos de salida:

- strABE: El arreglo de caracteres que contiene letras del abecedario

```
def ConvertirBinAbe(listArr: list, dicBinAbe: dict) -> str:
    strAbe = ""

    for elemento in listArr:
        strAbe = strAbe + dicBinAbe[elemento]

    return strAbe
```

Ilustración 9 – Función ConvertirBinAbe.

Finalmente, la última función que definimos es la de OperacionOrExclusiva la cual realiza la cual su nombre lo indica se encarga de realizar la operación lógica XOR a un arreglo de caracteres Binarios la cual se muestra en la *Ilustración 10*, sus parámetros den entrada son los siguientes.

Argumentos de entrada:

- listaLi: Es una lista de arreglos de números binarios la cual se muestra en el diagrama a bloques.
- listaFRi: Es la lista de arreglo de números binarios obtenida después de aplicar la función de Sustitución y de Permutación a un arreglo de caracteres del abecedario.

Y como argumentos de salida tenemos:

- str: La cual nos devuelve un arreglo de caracteres del abecedario como resultado de la operación XOR.

```python
def OperacionOrExclusiva(listaLi: list, listaFRi: list) -> str:
    listAux = []

    for posicion in range(len(listaLi)):

        arreBinLi = listaLi[posicion]
        arrgBinFRi = listaFRi[posicion]
        lisBinLi = []
        listBinFRi = []
        arregloAux = ""

        for elemnto in arreBinLi:
            lisBinLi.append(elemnto)

        for elemento in arrgBinFRi:
            listBinFRi.append(elemento)

        for iterador in range(len(lisBinLi)):
            valOpeOrExc = int(lisBinLi[iterador]) ^ int(listBinFRi[iterador])
            arregloAux += str(valOpeOrExc)

        listAux.append(AplicarMod27(strBin=arregloAux, dicNumBin=num_bin))

    return ConvertirBinAbe(listArr=listAux, dicBinAbe=bin_abe)
```

Ilustración 10 - Función OperacionOrExclusiva.

Implementación del programa

Por ultimo para finalizar declaramos una lista con las condiciones que nos dio el problema y el mensaje de texto plano que vamos a cifrar, por medio de un ciclo For iteramos dos veces y mandamos a llamar las funciones declaradas anteriormente, en la *Ilustración 11* se puede apreciar el orden en que se mandan a llamar dichas funciones.

```
lista8Caracteres = ["CRIP", "TOGR", "AFIA", "ESUN", "ACIE", "NCIA"]

print("\n**********************************************************")
print("Texto plano")
print(lista8Caracteres)
print("**********************************************************\n")

for i in range(2):

    FRi = Permutacion(Sustitucion(arr4Car=lista8Caracteres[1], dicAbeNum=abe_num, dicNumBin=num_bin))
    Li = ConvertirAbeBin(arreglo=lista8Caracteres[0], dicAbeNum=abe_num, dicNumBin=num_bin)
    LixFri = OperacionOrExclusiva(listaLi=Li, listaFRi=FRi)
    lista8Caracteres[0] = lista8Caracteres[1]
    lista8Caracteres[1] = LixFri

    FRi = Permutacion(Sustitucion(arr4Car=lista8Caracteres[3], dicAbeNum=abe_num, dicNumBin=num_bin))
    Li = ConvertirAbeBin(arreglo=lista8Caracteres[2], dicAbeNum=abe_num, dicNumBin=num_bin)
    LixFri = OperacionOrExclusiva(listaLi=Li, listaFRi=FRi)
    lista8Caracteres[2] = lista8Caracteres[3]
    lista8Caracteres[3] = LixFri

    FRi = Permutacion(Sustitucion(arr4Car=lista8Caracteres[5], dicAbeNum=abe_num, dicNumBin=num_bin))
    Li = ConvertirAbeBin(arreglo=lista8Caracteres[4], dicAbeNum=abe_num, dicNumBin=num_bin)
    LixFri = OperacionOrExclusiva(listaLi=Li, listaFRi=FRi)
    lista8Caracteres[4] = lista8Caracteres[5]
    lista8Caracteres[5] = LixFri

    print("\n**********************************************************")
    print(lista8Caracteres)
    print("**********************************************************\n")

print("\n**********************************************************")
print("Texto cifrado")
print(lista8Caracteres)
print("**********************************************************\n")
```

Ilustración 11 - Parte principal del código donde se mandan a llamar las funciones definidas anteriormente.

Resultados

Finalmente, en desplegamos nuestro programa en la consola de comando utilizando el comando py “nombre_programa.py” y así finalmente podemos ver como este nos muestra en pantalla el texto plano, el texto en la primera y segunda iteración del algoritmo, así como el texto cifrado final, esto se puede apreciar en la *Ilustración 12*.

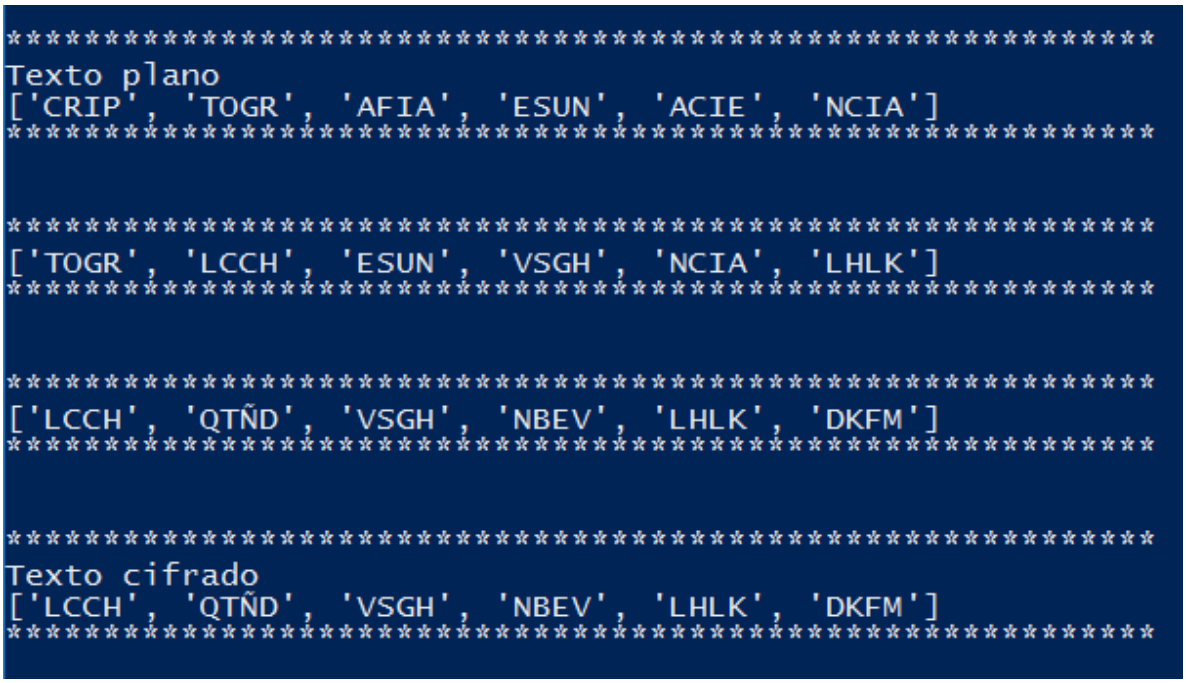


Ilustración 12 - Resultado en pantalla del algoritmo de cifrado de bloque.

Por último, se adjunta al link al repositorio donde se encuentra alojado el programa que se desarrolló a lo largo de este documento.

https://github.com/AlanCruzCh/Cifrado-de-Bloque

Concusiones

El desarrollo de este algoritmo ha sido un tanto complicado al inicio de entender ya que hay que realizar una serie de pasos que si bien está claramente definidos si no se pone la debida atención podría hacer que el algoritmo fallara, computacionalmente hablado su complejidad al menos con la solución que se presenta en este documento es de $O = n^3$ lo cual lo hace bastante deficiente si el tamaño del bloque aumenta y a pesar de eso al menos para casos prácticos pequeños a pesar de tener esa complejidad el programa da una respuesta bastante rápida.