



INSTITUTO POLITÉCNICO  
NACIONAL



ESCUELA SUPERIOR DE CÓMPUTO

CRYPTOGRAPHY

---

## Hill Cipher and Permutation Cipher

---

*Autores:*

González Núñez Daniel Adrián  
Hernández Castellanos César Uriel

*Docente:*

Dra. Sandra Diaz Santiago

**Ingeniería en Sistemas Computacionales**

8 de marzo de 2020

# 1. Introducción.

## Hill cipher

El cifrado de Hill(Hill cipher) es un algoritmo de criptosistema polialfabético que fue inventado en 1929 por Lester S. Hill, de ahí se desprende su nombre. EL algoritmo consiste en lo siguiente:

Sea  $m$  un entero positivo y definamos a  $(Z_{26})^m$  como nuestro conjunto de caracteres que forman parte del alfabeto. La idea de este algoritmo es la de generar  $m$  combinaciones lineales de los  $m$  caracteres pertenecientes al alfabeto que se encuentran en el mensaje plano a encriptar. Esto nos producirá  $m$  caracteres que de igual manera pertenecerán a nuestro alfabeto y que formarán al texto cifrado.

En este algoritmo se usan matrices cuadradas para generar las combinaciones lineales necesarias. Sea un elemento del texto plano  $x = (x_1, x_2)$  y un elemento de nuestro texto cifrado  $y = (y_1, y_2)$ . Podemos generar a  $y_1$  como una combinación lineal de  $x_1$  y  $x_2$  así como conseguir a  $y_2$  de la misma manera. Hay que recordar que todas las operaciones se deberán hacer modulo el tamaño de nuestro alfabeto, en este caso es modulo  $Z_{26}$ . Una combinación lineal quedaría de la forma:

$$\begin{aligned} y_1 &= (ax_1 + bx_2) \bmod N \\ y_2 &= (cx_1 + dx_2) \bmod N \end{aligned} \quad (1)$$

Esto se puede escribir de la siguiente manera como una matriz de coeficientes la cual deberá ser multiplicada por la izquierda por nuestro texto plano:

$$(y_1, y_2) = (x_1, x_2) \begin{pmatrix} a & b \\ c & d \end{pmatrix} \quad (2)$$

Para hallar el texto original a partir de la combinación  $y = (y_1, y_2)$  lo que debemos hacer será usar la inversa de la matriz de coeficientes que usamos como llave para generar la combinación lineal. Ahora, multiplicaremos con el texto cifrado por la izquierda a esta matriz inversa para así obtener el texto original.

$$(x_1, x_2) = (y_1, y_2) \begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1} \quad (3)$$

## Cifrado por permutacion

En criptografía, un cifrado por transposición es un tipo de cifrado en el que unidades de texto plano se cambian de posición siguiendo un esquema bien definido; las 'unidades de texto' pueden ser de una sola letra (el caso más común), pares de letras, tríos de letras, etc.

Este tipo de cifradores eran muy usados en la criptografía clásica y por tanto, al tener que hacer los cálculos por medios muy básicos.

Este tipo de algoritmos son de clave simétrica porque es necesario que tanto el que cifra como el que descifra sepan la misma clave para realizar su función.

## 2. Hill cipher

### 2.1. Descripción de la solución.

```
def readMatrixValues(matrixFile):
    sourceFile = matrixFile+".txt"
    values = open(sourceFile, 'r').read()
    a,b,c,d,e,f,g,h,i = values.split()
    matrix = np.array([[int(a),int(b),int(c)],[int(d),int(e),int(f)],[int(g),int(h),int(i)]])
    return matrix
```

Figura 1: Código fuente de la función 'testApp()'

En la figura *figura 1* mostramos el código de la función usada para poder leer los valores de la matriz definida en el archivo usado por el usuario.

El usuario tendrá que utilizar un archivo que contenga una matriz con un formato válido, previamente establecido, para que los valores se lean de manera correcta y puedan ser usados por el cifrador. La función leerá el texto y dividirá los valores por espacios y saltos de líneas para después crear una matriz con los valores enteros que el usuario haya proporcionado.

El formato que deberá ser usado para que funcione el cifrador es el siguiente:

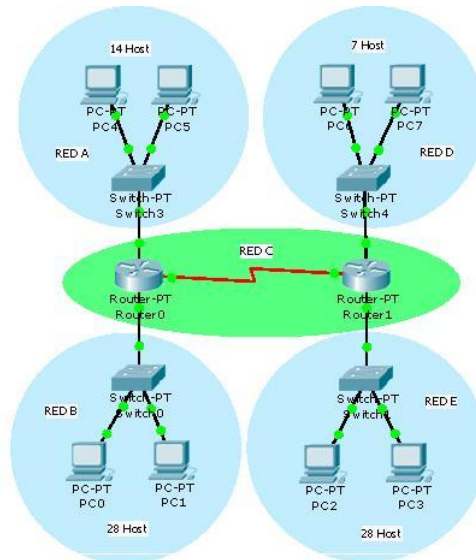


Figura 2: Formato válido para introducir una matriz de 3x3

```
def validateValues(matrix, fileSource, type_cf):
    errorType = 0
    factor = 0.001
    fileSize = int(os.path.getsize(fileSource+".txt"))*factor
    gcd = getEuclidean(getDeterminant(matrix))[0]
    if not(gcd == 1):
        errorType = 1
    if fileSize<5 and type_cf:
        errorType = 4
    return errorType
```

Figura 3: Función encargada validar los valores introducidos por el usuario

La función es la encargada de notificar los errores, derivados de el ingreso de datos por el usuario o por el tamaño del archivo que contiene el texto plano. La función simplemente retornar el error que se este cometiendo.

```

1 def encryptOrDecryptWord(originalWord, alphabet,matrix,type_cf):
2     cipherWord = ""
3
4     if type_cf == False:
5         matrix = getMatrixInverse(matrix)
6
7     temporaryList = list(originalWord)
8     listOfKeys = []
9
10    for i in range(len(temporaryList)):
11        temporaryString = str(temporaryList[i])
12        if temporaryString in alphabet.values():
13            lKey = [key for key, value in alphabet.items() if value == temporaryString][0]
14            listOfKeys.append(lKey)
15
16    faltantes = len(listOfKeys) % matrix_size
17    for i in range(faltantes):
18        listOfKeys.append(0)
19
20    for i in range(0,len(listOfKeys),matrix_size):
21        auxList = np.array([0,0,0]);
22        for j in range(matrix_size):
23            auxList[j] = listOfKeys[i+j]
24        newKeys = np.dot(auxList,matrix) % sizeAlphabet
25        for j in range(len(newKeys)):
26            cipherWord+=(alphabet[newKeys[j]])
27
28    return cipherWord

```

Figura 4: Función encargada de cifrar o descifrar el texto en un archivo implementada en Python

En la *figura* anterior se muestra la implementación en Python de la función. La función recibe como parámetros la palabra a descifrar( en este caso le mandamos el texto completo), el alfabeto que usaremos, la matriz correspondiente a la llave y una bandera que indica la operación a realizar, cifrado o descifrado.

Si la bandera indica que descifraremos la palabra entonces se saca la inversa de esa matriz para poder hacer el descifrado correctamente (líneas 4-5). Después, en el ciclo de las líneas 10-14, conseguiremos las llaves correspondientes a cada letra de la palabra a descifrar y las insertaremos en un arreglo.

En las líneas 16-18 lo que haremos será checar si el texto que estamos leyendo puede ser multiplicado por la matriz de tamaño 3x3 con la que contamos. Esto lo hacemos mediante el cálculo del tamaño del texto modulo 3, ya que nos interesa saber si las multiplicaciones se pueden hacer sin que haya problema. En el caso de que el modulo no sea igual a cero, lo que haremos será agregar unos caracteres extra hasta que el tamaño sea el adecuado para realizar la multiplicación de matrices.

En el siguiente ciclo (20-26) procederemos a seleccionar pedazos de tamaño 3 de nuestro arreglo de llaves obtenido a partir del arreglo original. Agarraremos tres llaves consecutivas y procederemos a multiplicar estas tres por nuestra matriz de tamaño 3x3 para obtener las otras tres nuevas llaves. Se debe tomar en consideración que todas estas operaciones se deben hacer aplicando modulo el tamaño de nuestro alfabeto, esto con el objetivo de conseguir llaves válidas. Sacaremos los caracteres correspondientes a cada llave y los concatenaremos a nuestro mensaje final.

Al final retornaremos el texto cifrado.

A continuación presentaremos las funciones usadas para poder hacer las operaciones de matrices correspondientes:

```

1  def getDeterminant(matrix):
2      return round(np.linalg.det(matrix) % sizeAlphabet)
3
4  def getAdjoint(matrix):
5      cofactor = np.linalg.inv(matrix).T * np.linalg.det(matrix)
6      aux = cofactor.transpose()
7      for i in range(matrix_size):
8          for j in range(matrix_size):
9              aux[i][j] = round(aux[i][j]%sizeAlphabet)
10     return aux
11
12  def getEuclidean(a):
13      tupleValues = extendedEuclideanA(a,sizeAlphabet)
14      return tupleValues[0],tupleValues[1]
15
16  def getMatrixInverse(matrix):
17      inverse = np.array([[ -1,-1,-1], [ -1,-1,-1], [ -1,-1,-1]])
18      determinant = getDeterminant(matrix)
19      gcd,multiplicativeInverse = getEuclidean(determinant)
20      if gcd == 1:
21          adjoint = getAdjoint(matrix)
22          inverse = (multiplicativeInverse*adjoint)%sizeAlphabet
23      return inverse

```

Figura 5: Función encargada de cifrar o descifrar una palabra

La función mas importante es la que podemos ver en las líneas 16-23, ya que esta la que usaremos para poder calcular la inversa de la matriz que usaremos para poder descifrar algún texto. Primero calcularemos el determinante de la matriz llave: Esto se hace mediante la primera función en la cual usamos una librería de NumPy para poder hacer el cálculo correspondiente (recordar que todo lo haremos modulo el tamaño de nuestro alfabeto).

Volviendo a la función principal, el siguiente paso es calcular el GCD y, en caso de que tuviera, el inverso multiplicativo del determinante modulo el tamaño del arreglo. Después, haremos una validación para observar si el GCD que obtuvimos es igual a 1, ya que esta es una condición necesaria y clave para poder obtener la inversa de cualquier matriz cuando trabajamos en un  $Z_n$  específico.

Si es que el GCD obtenido fue igual a 1, procederemos a calcular la matriz adjunta de nuestra matriz llave. Esto se hace en las líneas 4-10, en las cuales se harán uso de librerías de NumPy para: 1. Sacar la matriz de cofactores de la matriz que queremos y 2. Transponer esa matriz de cofactores para obtener como resultado a la matriz adjunta. Después de que hayamos obtenido la matriz adjunta, lo único que resta es multiplicar los valores de esta por el inverso multiplicativo del determinante modulo el tamaño del alfabeto, el cual ya habíamos calculado previamente junto con el GCD.

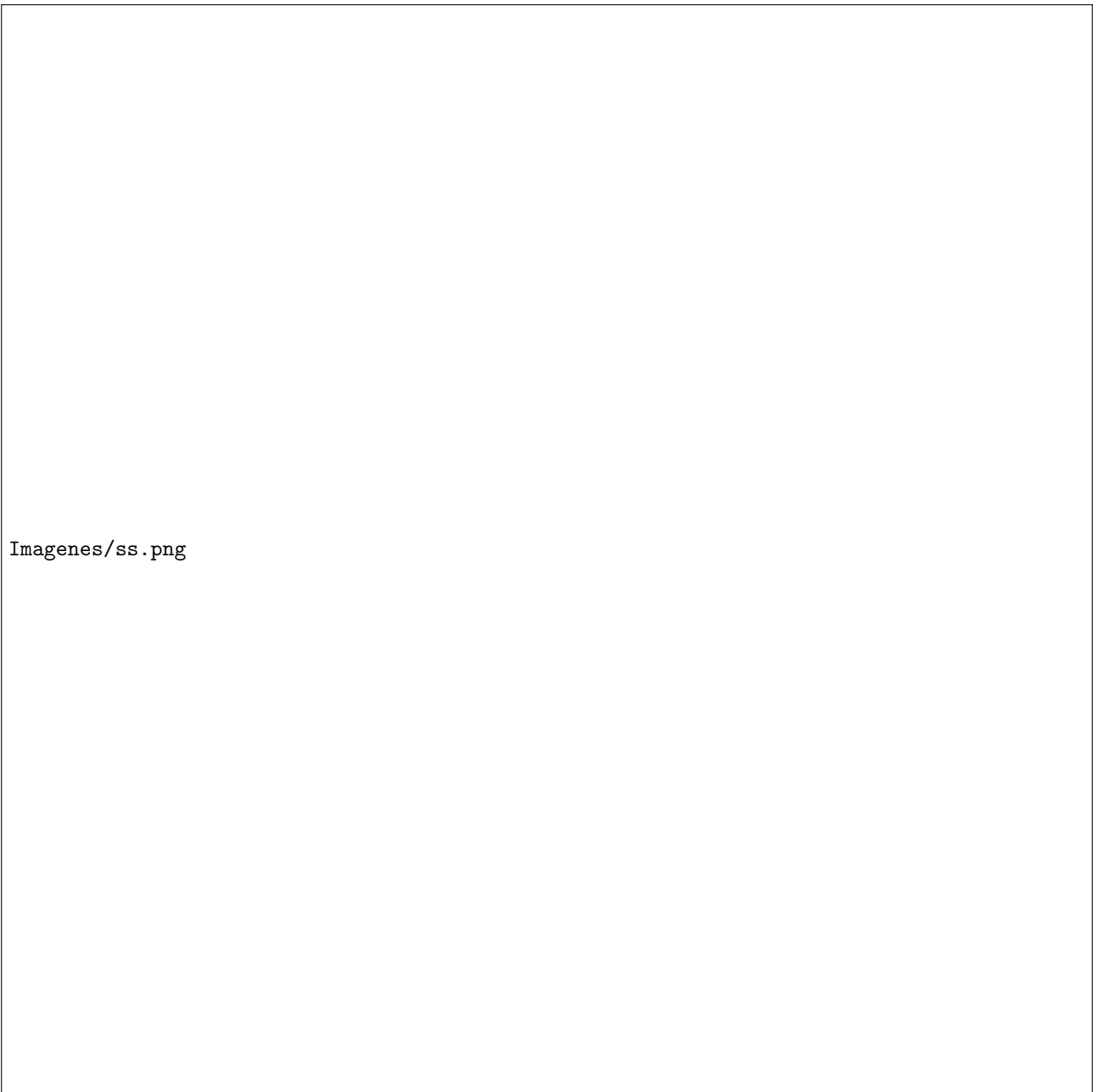
Al final, esta función nos retornará la inversa de la matriz llave que hayamos mandado como parámetro.

## 2.2. Excepciones.

En la tabla que se muestra a continuación se muestran los posibles errores que se puedan generar a la hora de la ejecución del programa con su respectiva descripción.

Tipo	Descripción
1	El gcd entre el determinante de la matriz y el tamaño del alfabeto es diferente a 1
4	El tamaño del archivo a cifrar tiene un tamaño menor a 5 kb

### 2.3. Output del programa



Imagenes/ss.png

Figura 6: Salida del programa que implementa el cifrado de Hil


### 3. Permutation Cipher

#### 3.1. Descripción de la solución.

```
def obtainDictionaryFromFile(permutationKeyFile,m):  
    f = open(permutationKeyFile,"r")  
    key = f.read()  
    auxList = re.findall(r'\b\d+\b', key)  
    listOne = []  
    temporaryDictionary = {}  
  
    for i in range(len(auxList)):  
        iElement = auxList[i]  
        if i >= m:  
            listOne.append(iElement)  
  
    for i in range(m):  
        temporaryDictionary[i] = listOne[i]  
  
    f.close()  
  
    return temporaryDictionary
```

Figura 7: Función que transforma el contenido de un archivo (llave) en un diccionario

La figura anterior es posible observar la implementación de la función que obtiene una llave de un fichero, la cual tiene el siguiente formato



Imagenes/key.png

Figura 8: Llave en un fichero

La función se recibe como parámetros el archivo que contiene la llave y el número de permutaciones.

Primeramente se abre el fichero en modo lectura con el fin de obtener su contenido, el cual será como entrada a la expresión regular que filtrará únicamente números enteros los cuales son almacenados en una lista.

Posteriormente iteramos la lista, con el fin de almacenar los elementos que sean mayores que  $m$ , ya que estos son los que nos importan.

Finalmente se transforma la lista a un diccionario para poder retornarlo.



```
def obtainAsciiAlphabet():
    sizeAlphabet      = 94
    specialCases      = 6
    temporaryList = list(string.printable)
    temporaryDictionary = {}

    for i in range(specialCases):
        temporaryList.pop()

    for i in range(sizeAlphabet):
        temporaryDictionary[i] = temporaryList[i]

    return temporaryDictionary
```

Figura 9: Función que obtiene el alfabeto

La función anterior es la encargada de generar el alfabeto que contiene los caracteres imprimibles del ascii en manera de diccionario, por lo que se aprovechan el módulo string de python para poder obtenerlos

```
def filterText(plainText, alphabet):
    finalString      = ""
    temporaryList     = list(plainText)

    for i in range(len(temporaryList)):
        if temporaryList[i] in alphabet.values():
            finalString+=temporaryList[i]

    return finalString
```

Figura 10: Función que verifica que el texto plano se encuentre en el alfabeto

La función filtrar texto nos auxilia en desechar todo aquello que no se encuentre en el alfabeto, con la finalidad de evitar errores a futuro, esto se hace por medio de un ciclo que recorra el texto plano y verificar que cada caractere se encuentre en el alfabeto, finalmente se retorna la cadena filtrada

```

def dividePlainTextIntoCharacters(text,m):
    temporaryList = list(text)
    finallist = []
    sizeText = len(temporaryList)
    residue = sizeText % m
    temporaryNumber = 0
    temporaryString = ""

    for i in range(sizeText):
        temporaryNumber+=1
        temporaryString += str(temporaryList[i])

        if temporaryNumber == m:
            finallist.append(temporaryString)
            temporaryString = ""
            temporaryNumber = 0

    if residue!=0:
        surplusStringSize = len(temporaryString)

        for i in range(m-surplusStringSize):
            temporaryString+=extraCharacter

        finallist.append(temporaryString)

    return finallist

```

Figura 11: Función que obtiene bloques de longitud m

La función mostrada en la figura anterior tiene como misión tokenizar el texto plano en cadenas de longitud m.

Esto es posible casteando el texto a lista, lo que nos retornará una lista donde cada elemento es un carácter del texto plano, para posteriormente iterar sobre la lista y verificar en cada iteración se aumenta una variable auxiliar con el fin de conocer el momento en el que se ha alcanzado una longitud m, con lo que obtenemos los tokens de longitud m.

Finalmente se trata el caso especial cuando la longitud del mensaje no es múltiplo del número de permutaciones, por lo que a estos espacios que quedan vacíos se les asigna un carácter especial

```

def obtainInversePermutation(dictionaryKey):
    return OrderedDict(sorted(dictionaryKey.items(), key=lambda x: x[1]))

```

Figura 12: Función que ordena un diccionario tomando como criterio sus valores

La función obtener permutación inversa, obtiene un diccionario que se encuentra ordenado con base en sus valores, esto se hace aprovechando la programación funcional, la cual nos ayuda a obtener un código más rápido de ejecutar.

```
def obtainRandomPermutation(m):
    temporaryDictionary = {}
    piList = [i for i in range(m)]
    random.shuffle(piList)
    savePermutationInFile(piList)

    for i in range(m):
        temporaryDictionary[i] = piList[i]

    inversePermutation = obtainInversePermutation(temporaryDictionary)

    return inversePermutation
```

Figura 13: Función que obtiene una permutación aleatoria

la función obtener permutación aleatoria recibe como parámetro el número de permutaciones.

Inicialmente se inicializa una lista con valores del 0 a m, para posteriormente desordenarlos de manera aleatoria, esta será la llave que usaremos para cifrar, la cual se almacenará un archivo de texto.

Finalmente se obtiene un diccionario a partir de estos datos y se retorna

```
def encryptPlainText(plainText,m):
    temporaryList = dividePlainTextIntoCharacters(plainText,m)
    temporaryDictionary = obtainRandomPermutation(m)
    finalCipherText = ""

    for i in range(len(temporaryList)):
        iList = list(temporaryList[i])
        iCipherText = ""
        for j in range(len(iList)):
            newIndex = temporaryDictionary[j]
            iCipherText+=str(iList[newIndex])
        finalCipherText+=iCipherText

    return finalCipherText
```

Figura 14: Función que cifra el texto plano

La función cifrar texto plano recibe como parámetro el texto plano y el número de permutaciones, para después mandar a llamar la función que nos divide el texto plano en bloques.

Como siguiente paso se obtiene una permutación aleatoria, para después iterar sobre cada bloques y obtener su respectivo texto cifrado.

La función que se encuentra a continuación se encarga de descifrar, para esto se reciben como parámetros el texto cifrado, el número de permutaciones y el archivo que contiene la llave.

En primer lugar se manda a llamar a la función que obtiene el diccionario a partir de un fichero, para que este mismo pase como parámetro a la función que obtiene la permutación inversa, posteriormente se obtiene una lista que contiene los bloques a cifrar, se itera en cada bloque y se obtiene de esa manera el texto plano.

```
def decryptPlainText(cipherText, m, permutationKeyFile):
    temporaryDictionary = obtainDictionaryFromFile(permutationKeyFile,m)
    inversePermutation = {val:key for (key, val) in temporaryDictionary.items()}
    temporaryList = dividePlainTextIntoCharacters(cipherText,m)
    inversePermutation = obtainInversePermutation(inversePermutation)
    finalDecipherText = ""

    for i in range(len(temporaryList)):
        idecipherText = ""
        iList = list(temporaryList[i])
        for j in range(len(iList)):
            newIndex = int(inversePermutation[str(j)])
            idecipherText+=str(iList[newIndex])

        finalDecipherText+=idecipherText

    return finalDecipherText
```

Figura 15: Función encargada de descifrar el texto plano

La función que se muestra abajo tiene como tarea escribir en un fichero lo que se le pase como parámetro

```
def saveText(text, fileSource):
    f = open(fileSource,"w")
    f.write(text)
    f.close
```

Figura 16: Texto cifrado

```

1  def testApp():
2      command = "clear"
3      subprocess.call(command, shell=True)
4      alphabet = obtainAsciiAlphabet()
5      option = input("1: Encrypt \n2: Decrypt \n3: Exit \nInput: ")
6
7      if option == 1:
8          subprocess.call(command, shell=True)
9          fileSource = raw_input("Enter the file which contains your plain text: ")
10         try:
11             m = input("Enter the number of permutations: ")
12         except NameError:
13             print("Error 1F has been made, see the documentation")
14             sys.exit()
15
16         errorType = validateEncryptValues(fileSource,m)
17
18         if errorType == "W":
19             subprocess.call(command, shell=True)
20             plainText = obtainTextFromFile(fileSource)
21             plainText = filterText(plainText,alphabet)
22             cipherText = encryptPlainText(plainText,m)
23             print("Successful encryption!\nYour key has been saved in the file
24                 ↪ "+permutationKeyFile)
25             saveText(cipherText,cipherTextFile)
26             print("Your cipher text has been saved in the file "+cipherTextFile)
27         else:
28             print("Error "+errorType+" has been made, see the documentation")
29             sys.exit()
30
31     elif option ==2:
32         subprocess.call(command, shell=True)
33         cipherTextFile_ = raw_input("Enter the file which contains the cipher text: ")
34         permutationKeyFile_ = raw_input("Enter the file which contains the permutation key:
35             ↪ ")
36
37         try:
38             m = input("Enter the number of permutations: ")
39         except NameError:
40             print("Error 2H has been made, see the documentation")
41
42         errorType = validateDecryptValues(cipherTextFile_,permutationKeyFile_,m)
43
44         if errorType == "W":
45             cipherText = obtainTextFromFile(cipherTextFile_)
46             decipherText = decryptPlainText(cipherText,m,permutationKeyFile_)
47             saveText(decipherText,decipherTextFile)
48             subprocess.call(command, shell=True)
49             print("Successful decryption!\nYour decipher text has been saved in the file
50                 ↪ "+decipherTextFile)
51         else:
52             print("Error "+errorType+" has been made, see the documentation")
53             sys.exit()
54     else:
55         sys.exit()

```

Figura 17: Función principal del programa

La función probar aplicación es la principal, ya que en esta se solicita al usuario si desea cifrar o descifrar, en caso que la opción sea cifrar se solicita el archivo que contiene el texto plano, el número de permutaciones y se manda a llamar la función que valida todos los posibles errores, en caso de al menos un error de estos ocurra se finalizará el programa en caso contrario se obtiene el texto plano del archivo y se filtra de acuerdo al alfabeto, para posteriormente cifrar el texto y almacenarlo en un fichero.

En el caso de que se opte por la opción de descifrar se solicita el archivo que contiene el texto cifrado y el archivo que contiene la llave, para posteriormente descifrar el texto y almacenarlo en un fichero.

```
def obtainFileSize(fileSource):  
    factor = 0.001  
    fileSize = int(os.path.getsize(fileSource))*factor  
    return fileSize
```

Figura 18: Función que obtiene el tamaño de un archivo

La figura anterior muestra la función que obtiene el tamaño de un archivo en kb, para esto se aprovecha el módulo os de python

```
def validateDecryptValues(cipherTextFile, permutationKeyFile, m):  
    errorType = "W"  
    fileExist0 = os.path.isfile(cipherTextFile)  
    fileExistT = os.path.isfile(permutationKeyFile)  
  
    if fileExist0 == False and fileExistT == False and m < 0:  
        errorType = "2A"  
    elif fileExist0 == False and fileExistT == False:  
        errorType = "2B"  
    elif fileExist0 == False and m < 0:  
        errorType = "2C"  
    elif fileExistT == False and m < 0:  
        errorType = "2D"  
    elif fileExist0 == False:  
        errorType = "2E"  
    elif fileExistT == False:  
        errorType = "2F"  
    elif m < 0:  
        errorType = "2G"  
  
    return errorType
```

Figura 19: Manejador de errores en el descifrado

La función que valida los datos de entrada del descifrado recibe como parámetros el archivo que contiene el texto cifrado, el archivo que contiene la llave y el número de permutaciones, con el fin de validar en la función de que existan los archivos y de que el número de permutaciones sea mayor que cero

```

def validateEncryptValues(fileSource,m):
    errorType = "W"
    fileExist = os.path.isfile(fileSource)

    if fileExist == False and m < 0:
        errorType = "1A"
    elif fileExist ==False:
        errorType = "1B"
    elif obtainFileSize(fileSource) < 5 and m < 0:
        errorType = "1C"
    elif obtainFileSize(fileSource) < 5:
        errorType = "1D"
    elif m < 0:
        errorType = "1E"

    return errorType

```

Figura 20: Manejador de errores del cifrado

En la figura anterior se valida de que el archivo que contiene el texto plano exista y que el número de permutaciones sea positivo.

Tanto en el manejador de errores del cifrado y descifrado proporcionan un tipo de error de los cuales se explicaran más adelante.

```

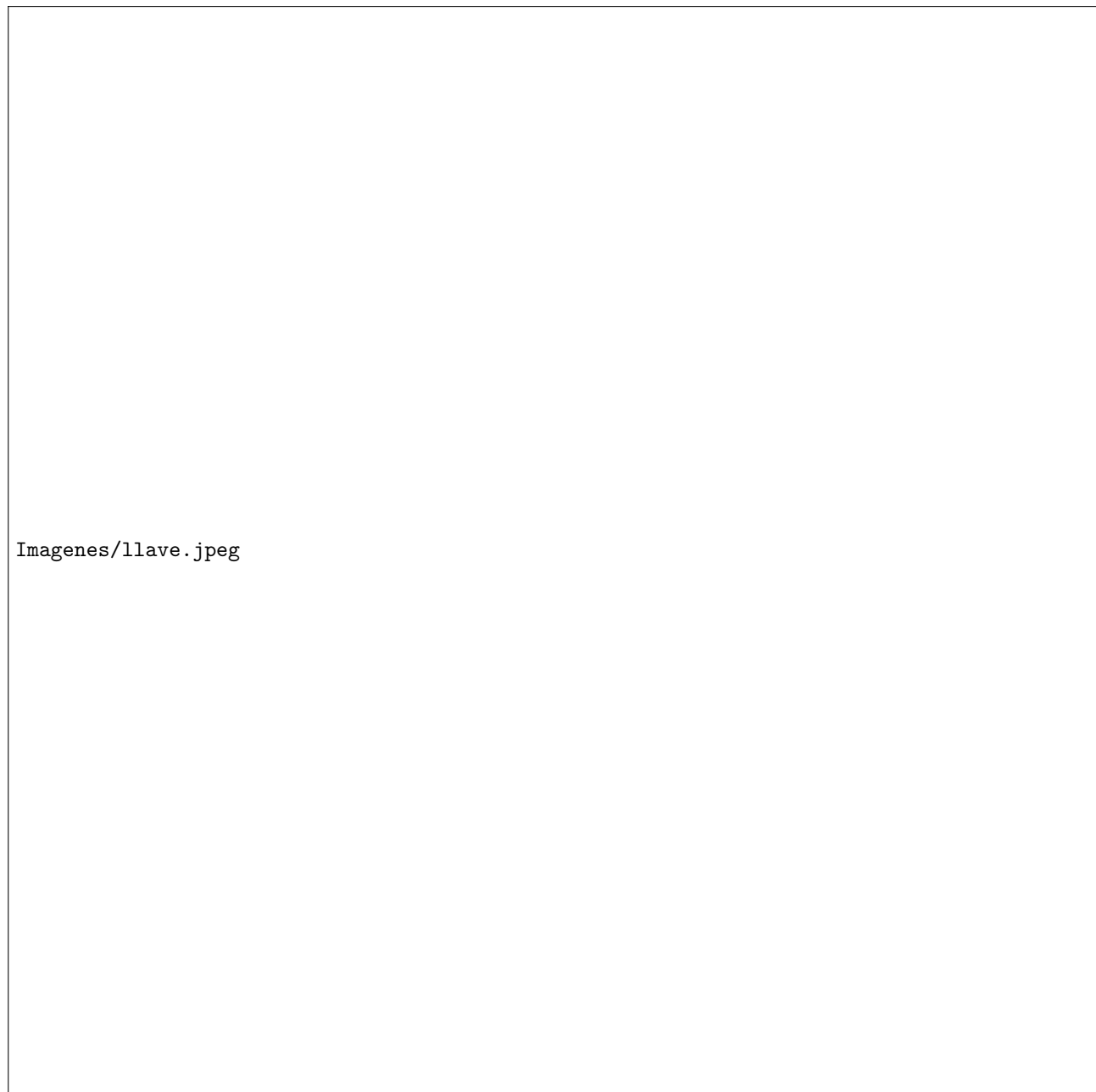
def savePermutationInFile(permutationList):
    size = len(permutationList)
    f = open(permutationKeyFile,"w")
    f.write("x |")
    f.write(str([i for i in range(size)]).replace("[","").replace("]", ""))
    f.write("|")
    f.write("pi |")
    f.write(str(permutationList).replace("[","").replace("]", ""))
    f.write("|")
    f.close()

```

Figura 21: Texto cifrado

La función anterior es la encargada de almacenar la llave en un fichero

### 3.2. Output del programa



Imagenes/llave.jpeg

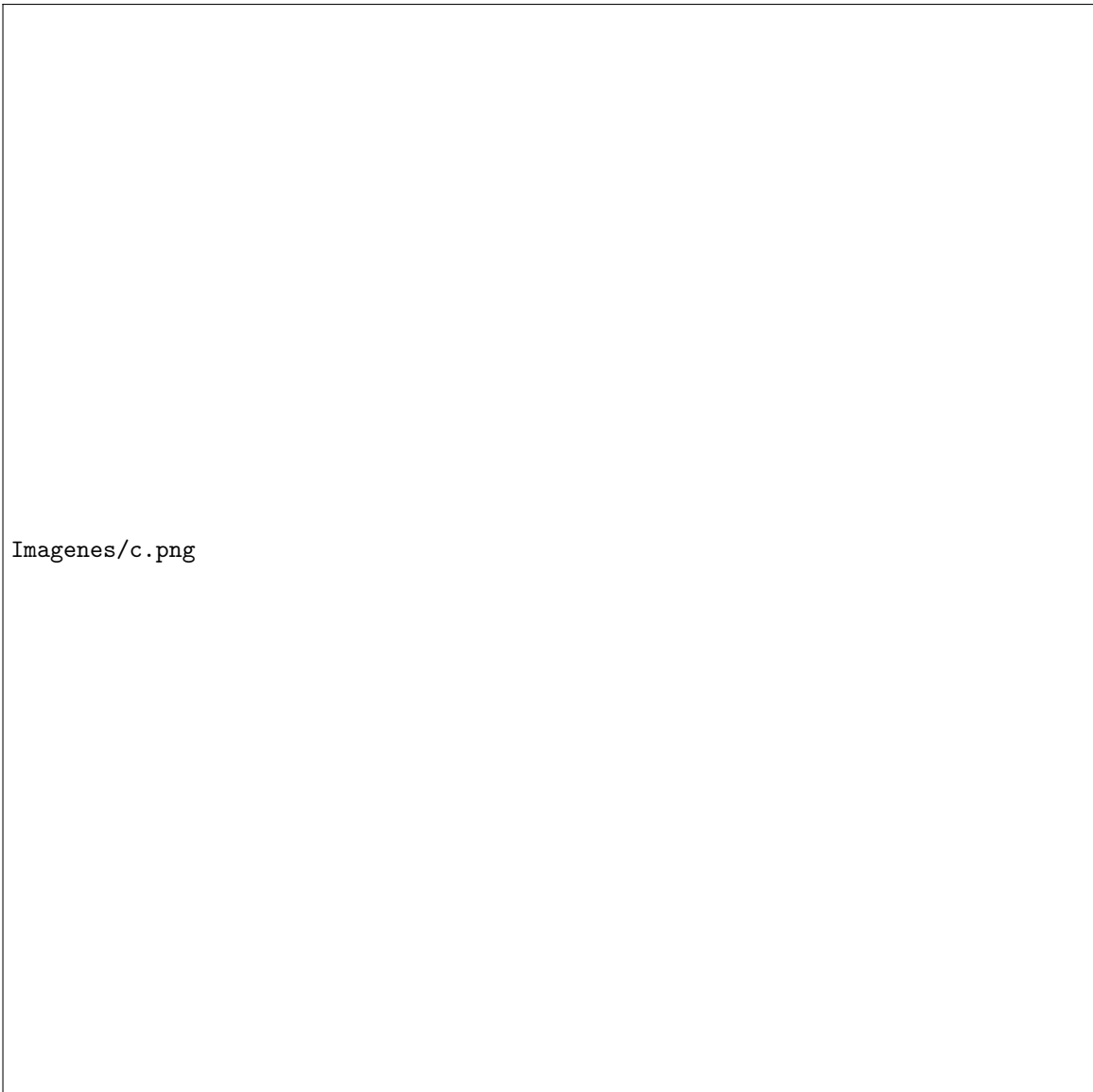
Figura 22: Salida del programa que implementa el cifrado por permutación

### 3.3. Excepciones.


En la tabla que se muestra a continuación se muestran los posibles errores que se puedan generar a la hora de la ejecución del programa con su respectiva descripción.



### 3.3.1. Excepciones para el cifrado



### 3.3.2. Excepciones para el descifrado



Imagenes/d.png

## 4. Conclusión

El surgimiento de nuevas redes de comunicación en lo particular el internet, abrió diferentes posibilidades para el intercambio de información, con lo que de manera casi proporcional aumentaron las amenazas de la seguridad de los canales de información. Es necesario entonces, crear diferentes mecanismos que nos "garanticen" la confidencialidad y autenticidad de los datos.

En el presente documento se revisa hill cipher y el cifrado por permutaciones, el primero hace uso de fundamentos de álgebra lineal que nos auxilia como a cifrar y descifrar, además de que es un cifrado de sustitución poligráfica, para su implementación se aprovecharon los diferentes módulos de python, por lo que no hubo ninguna dificultad en su implementación.

El segundo método que se revisa es el cifrado por permutación el cual es un tipo de cifrado en el que unidades de texto plano se cambian de posición siguiendo un esquema bien definido, este cifrador es muy usado en la criptografía clásica y por tanto hace uso de cálculos muy básicos. Las dificultades que se presentaron en el momento de la implementación de este cifrador surgió de decidir como estructurar un diccionario que almacenara la llave, por lo que se tuvo que se tenía que implementar un algoritmo que ordenara un diccionario de acuerdo

a sus valores lo cual provocó bastantes problemas, pero finalmente se pudo resolver de manera exitosa.

