Instituto Politécnico Nacional

Escuela Superior de Cómputo

Hill Cipher

Group: 3CM15

Student.

Maximiliano Cazares Martínez

How to run the program.

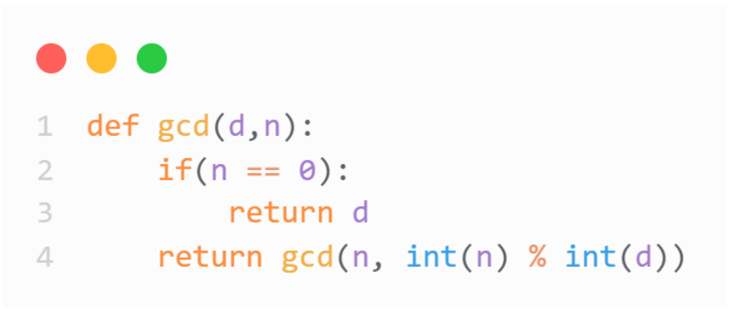
The program called Hill Cipher.py is a python file an it can be run in any computer with a python 3 interpreter. Additionally, it must be installing the package NumPy. If it is not installed, you can type the command "py -m pip install NumPy" on the terminal.

Organization.

The python file is organized in 7 functions. StoreKey and GetKey are two new functions that allow the file management for store the key.

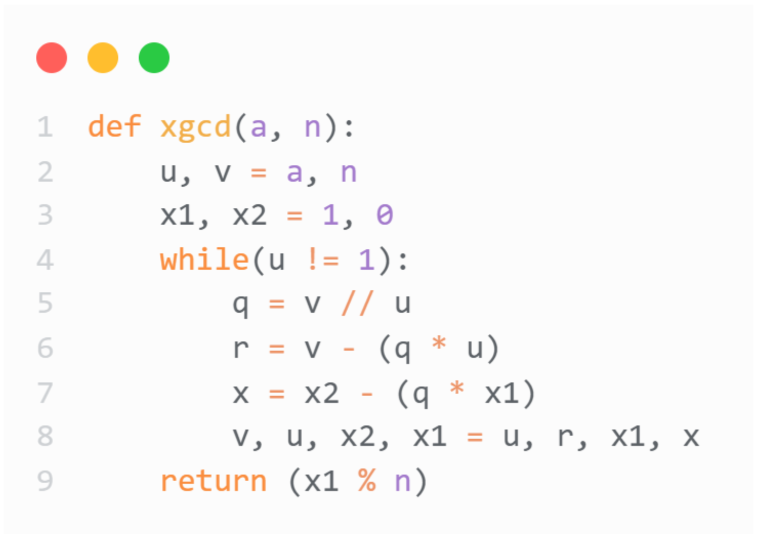
The functions gcd and xgcd were not modified.

- **Gcd:** This functions compute in an efficient way the greatest common divisor of two integer numbers.



```python
def gcd(d,n):
    if(n == 0):
        return d
    return gcd(n, int(n) % int(d))
```

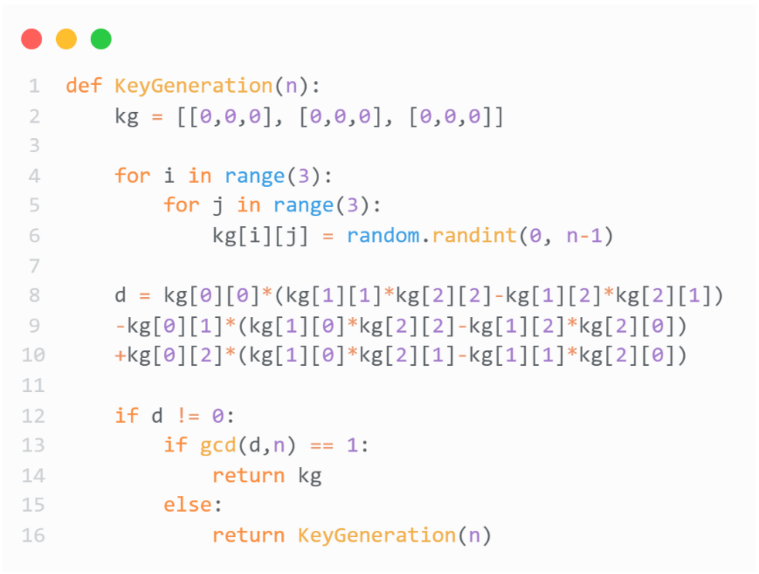
- **Xgcd:** This function is the Extended Euclidian Algorithm which one allow to compute the modular multiplicative inverse when two integer numbers are coprime.



```python
def xgcd(a, n):
    u, v = a, n
    x1, x2 = 1, 0
    while(u != 1):
        q = v // u
        r = v - (q * u)
        x = x2 - (q * x1)
        v, u, x2, x1 = u, r, x1, x
    return (x1 % n)
```

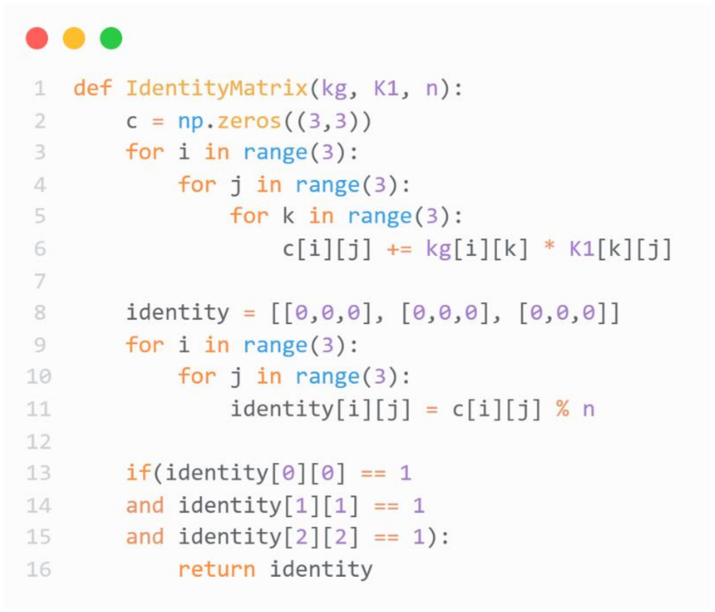
Finally, the remaining three functions were modified with respect to those previously sent.

- **KeyGeneration:** In the previous version we generate the key with functions of NumPy module but now, we generate it with loops, that's because the arithmetic operations are more precise in this way.
  The key is generated with random integer numbers between zero and the size of the alphabet minus one. Then it's verified that the determinant of the matrix is non-zero and the greatest common divisor of determinant and the size of the alphabet is equals one. It this conditional is true, the key is returned and in case it's not, the key is computed again.



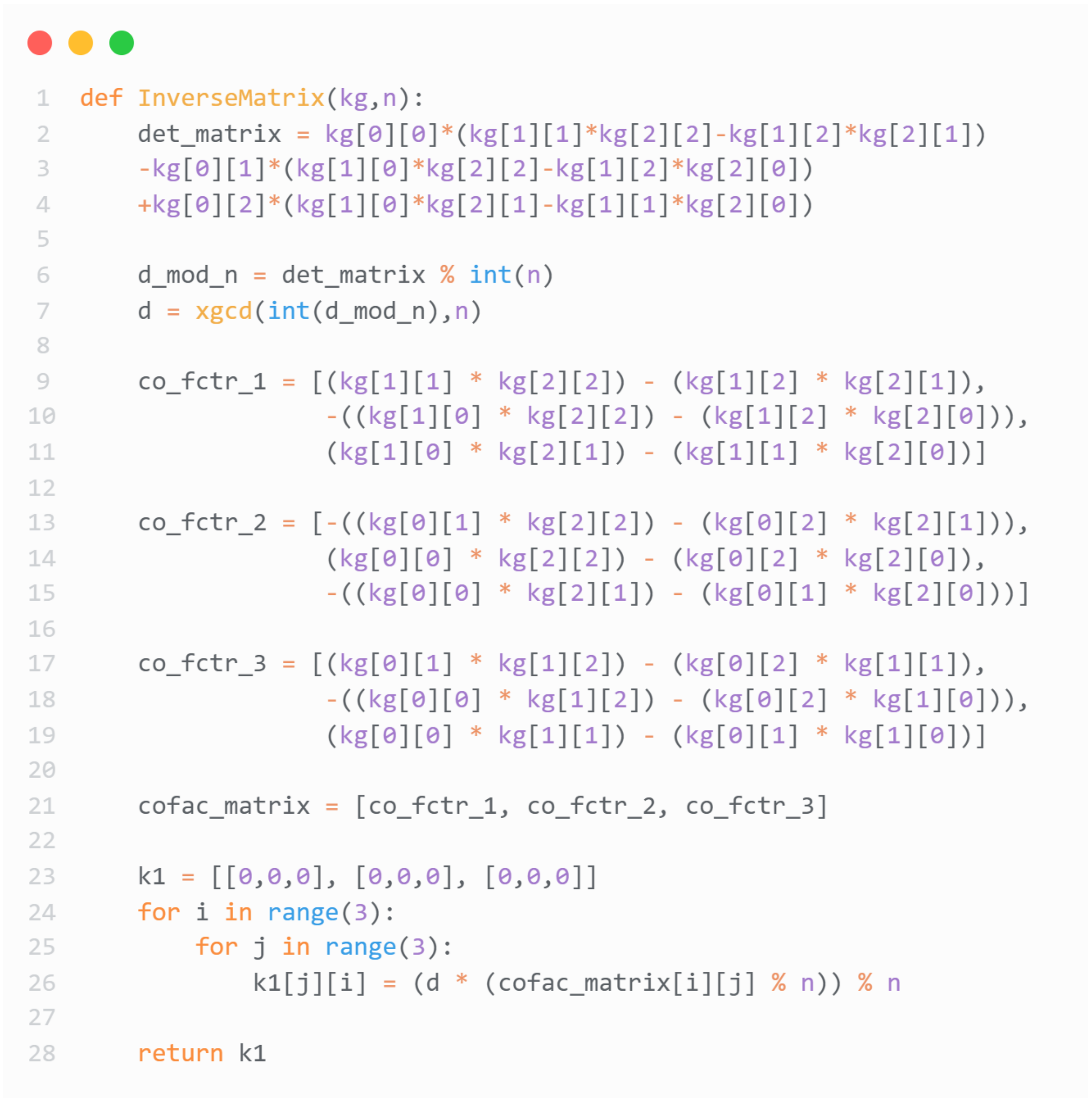
```python
def KeyGeneration(n):
    kg = [[0,0,0], [0,0,0], [0,0,0]]

    for i in range(3):
        for j in range(3):
            kg[i][j] = random.randint(0, n-1)

    d = kg[0][0]*(kg[1][1]*kg[2][2]-kg[1][2]*kg[2][1])
    -kg[0][1]*(kg[1][0]*kg[2][2]-kg[1][2]*kg[2][0])
    +kg[0][2]*(kg[1][0]*kg[2][1]-kg[1][1]*kg[2][0])

    if d != 0:
        if gcd(d,n) == 1:
            return kg
        else:
            return KeyGeneration(n)
```

- **IdentityMatrix:** It was only added a module operation in every element of the matrix product.
  We multiply the key with its inverse and we get the module n of every element of this product.



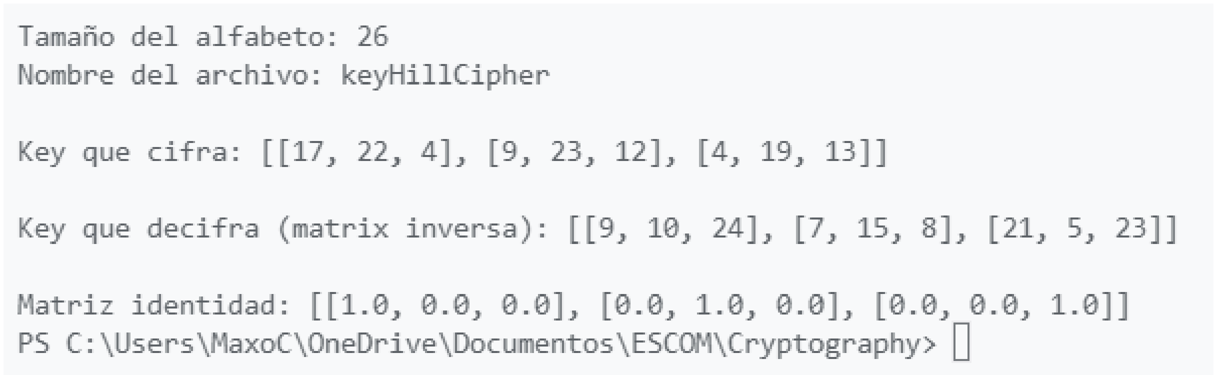
```python
def IdentityMatrix(kg, K1, n):
    c = np.zeros((3,3))
    for i in range(3):
        for j in range(3):
            for k in range(3):
                c[i][j] += kg[i][k] * K1[k][j]

    identity = [[0,0,0], [0,0,0], [0,0,0]]
    for i in range(3):
        for j in range(3):
            identity[i][j] = c[i][j] % n

    if(identity[0][0] == 1
    and identity[1][1] == 1
    and identity[2][2] == 1):
        return identity
```

- **InverseMatrix:** The modifications on this function consist in the way how to compute modular adjoint matrix of the cofactor matrix of the previously generated key. First, the determinant of the key and the modular multiplicative inverse are computed. Second, the cofactor matrix of the key is calculated. Third, we get the module n of the multiplication of the determinant of the key by the module n of every element in the cofactor matrix. Finally, we get the transposed matrix also called the inverse matrix.

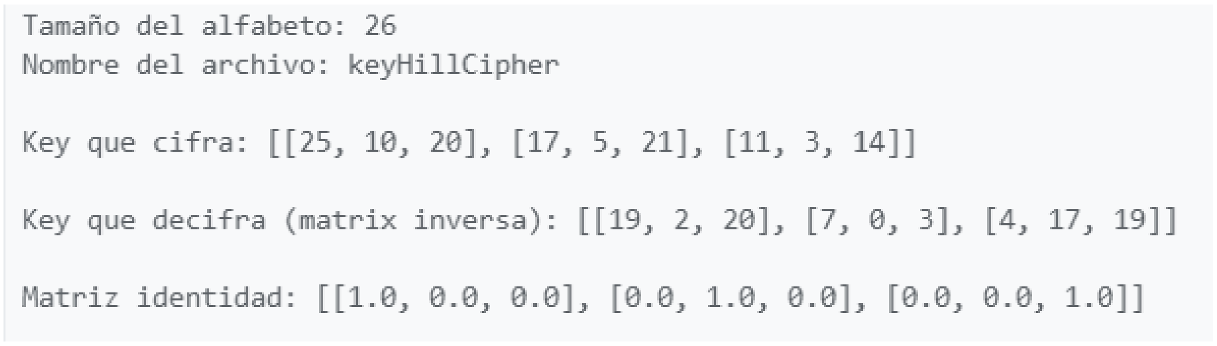


```python
def InverseMatrix(kg,n):
    det_matrix = kg[0][0]*(kg[1][1]*kg[2][2]-kg[1][2]*kg[2][1])
    -kg[0][1]*(kg[1][0]*kg[2][2]-kg[1][2]*kg[2][0])
    +kg[0][2]*(kg[1][0]*kg[2][1]-kg[1][1]*kg[2][0])

    d_mod_n = det_matrix % int(n)
    d = xgcd(int(d_mod_n),n)

    co_fctr_1 = [(kg[1][1] * kg[2][2]) - (kg[1][2] * kg[2][1]),
                 -((kg[1][0] * kg[2][2]) - (kg[1][2] * kg[2][0])),
                 (kg[1][0] * kg[2][1]) - (kg[1][1] * kg[2][0])]

    co_fctr_2 = [-((kg[0][1] * kg[2][2]) - (kg[0][2] * kg[2][1])),
                 (kg[0][0] * kg[2][2]) - (kg[0][2] * kg[2][0]),
                 -((kg[0][0] * kg[2][1]) - (kg[0][1] * kg[2][0]))]

    co_fctr_3 = [(kg[0][1] * kg[1][2]) - (kg[0][2] * kg[1][1]),
                 -((kg[0][0] * kg[1][2]) - (kg[0][2] * kg[1][0])),
                 (kg[0][0] * kg[1][1]) - (kg[0][1] * kg[1][0])]

    cofac_matrix = [co_fctr_1, co_fctr_2, co_fctr_3]

    k1 = [[0,0,0], [0,0,0], [0,0,0]]
    for i in range(3):
        for j in range(3):
            k1[j][i] = (d * (cofac_matrix[i][j] % n)) % n

    return k1
```

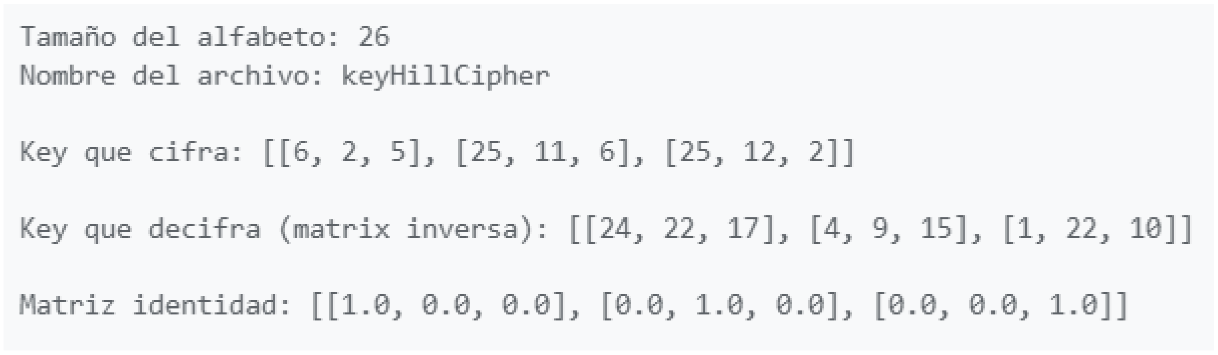
Runs.



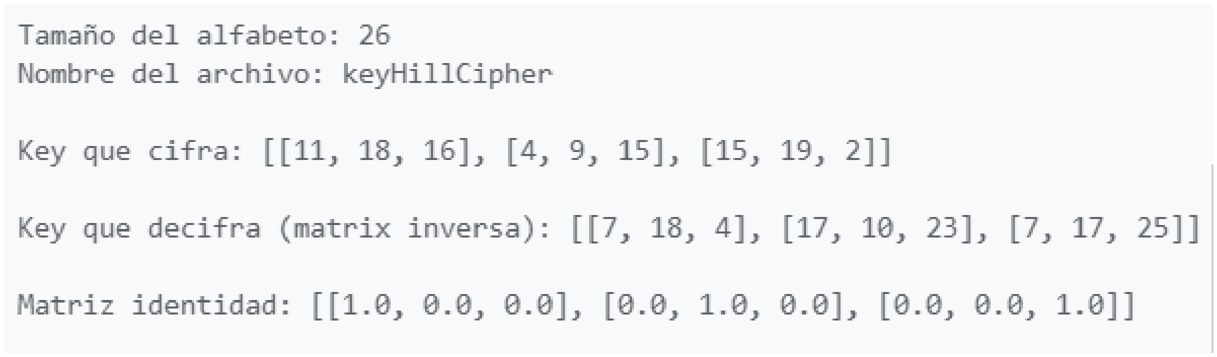
```
Tamaño del alfabeto: 26
Nombre del archivo: keyHillCipher

Key que cifra: [[17, 22, 4], [9, 23, 12], [4, 19, 13]]

Key que decifra (matrix inversa): [[9, 10, 24], [7, 15, 8], [21, 5, 23]]

Matriz identidad: [[1.0, 0.0, 0.0], [0.0, 1.0, 0.0], [0.0, 0.0, 1.0]]
PS C:\Users\MaxoC\OneDrive\Documentos\ESCOM\Cryptography>
```



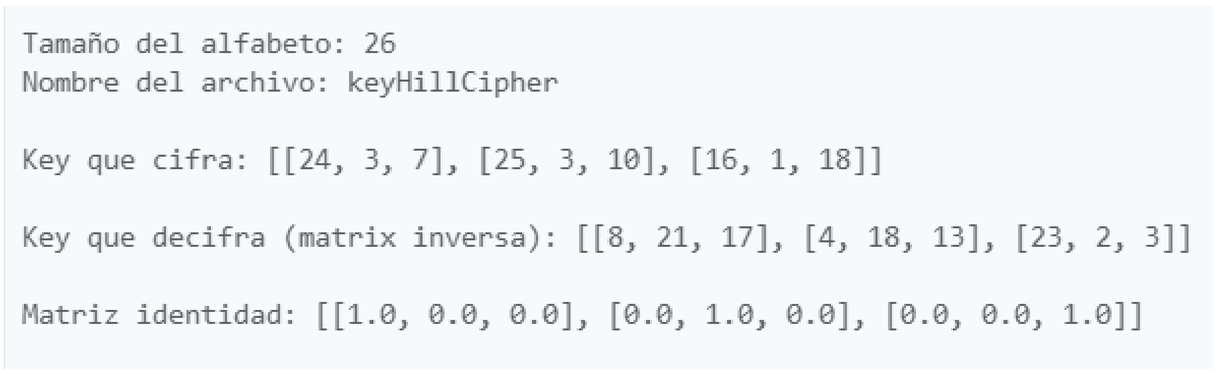
```
Tamaño del alfabeto: 26
Nombre del archivo: keyHillCipher

Key que cifra: [[25, 10, 20], [17, 5, 21], [11, 3, 14]]

Key que decifra (matrix inversa): [[19, 2, 20], [7, 0, 3], [4, 17, 19]]

Matriz identidad: [[1.0, 0.0, 0.0], [0.0, 1.0, 0.0], [0.0, 0.0, 1.0]]
```



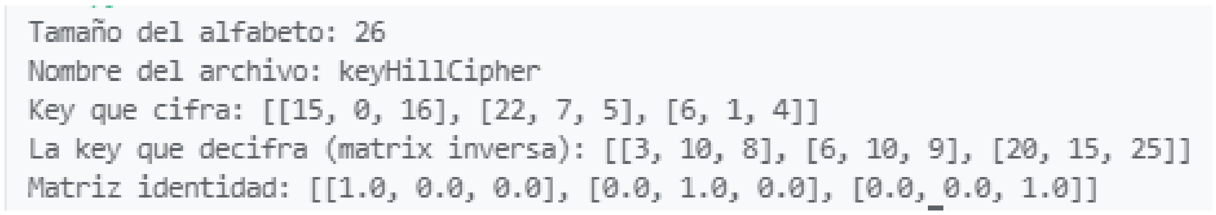
```
Tamaño del alfabeto: 26
Nombre del archivo: keyHillCipher

Key que cifra: [[6, 2, 5], [25, 11, 6], [25, 12, 2]]

Key que decifra (matrix inversa): [[24, 22, 17], [4, 9, 15], [1, 22, 10]]

Matriz identidad: [[1.0, 0.0, 0.0], [0.0, 1.0, 0.0], [0.0, 0.0, 1.0]]
```



```
Tamaño del alfabeto: 26
Nombre del archivo: keyHillCipher

Key que cifra: [[11, 18, 16], [4, 9, 15], [15, 19, 2]]

Key que decifra (matrix inversa): [[7, 18, 4], [17, 10, 23], [7, 17, 25]]

Matriz identidad: [[1.0, 0.0, 0.0], [0.0, 1.0, 0.0], [0.0, 0.0, 1.0]]
```



```
Tamaño del alfabeto: 26
Nombre del archivo: keyHillCipher

Key que cifra: [[24, 3, 7], [25, 3, 10], [16, 1, 18]]

Key que decifra (matrix inversa): [[8, 21, 17], [4, 18, 13], [23, 2, 3]]

Matriz identidad: [[1.0, 0.0, 0.0], [0.0, 1.0, 0.0], [0.0, 0.0, 1.0]]
```



```
Tamaño del alfabeto: 26
Nombre del archivo: keyHillCipher
Key que cifra: [[15, 0, 16], [22, 7, 5], [6, 1, 4]]
La key que decifra (matrix inversa): [[3, 10, 8], [6, 10, 9], [20, 15, 25]]
Matriz identidad: [[1.0, 0.0, 0.0], [0.0, 1.0, 0.0], [0.0, 0.0, 1.0]]
```