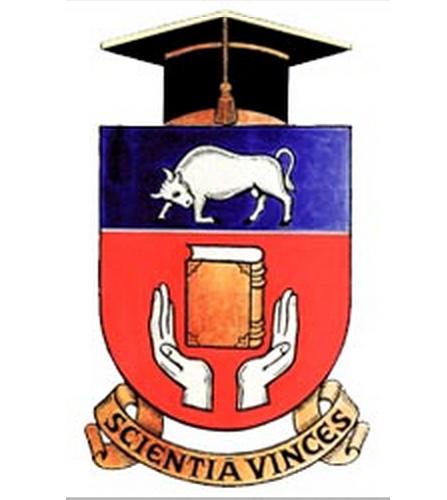
Universitatea de Stat din Tiraspol

Facultatea de Fizica Matematica si Informatica



Cifrul Hill

**Elaborat de**: Dordea Pavel, student gr.3i

**Verificat**: Globa Angela

**Chișinău, 2021**

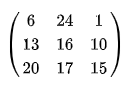
**Sistemul Hill**

***#Despre***

În criptografia clasică, **cifrul Hill** este un cifru al substituției poligrafic bazat pe algebră lineară. Inventat de către Lester S. Hill în 1929, a fost primul cifru poligrafic în care era practic posibil să se opereze cu mai mult de trei simboluri deodată. Pentru a înțelege discuția următoare sunt necesare cunoștințe de teoria matricelor.

***#Exemplu***

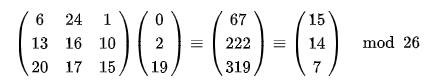
Fiecare literă este tratată ca o cifră din baza 26: A = 0, B =1 ș.a.m.d. Un bloc de *n* litere este considerat ca un vector cu n dimensiuni, și multiplicat cu o matrice n × n, modulo 26. Componentele matricei reprezintă cheia, care trebuie alese aleatoriu astfel încât matricea să fie inversabilă în {\displaystyle \mathbb {Z} \_{26}^{n}} (pentru a asigura posibilitatea decriptării). Considerăm mesajul 'ACT', și cheia de mai jos (sau GYBNQKURP în litere):



Deoarece 'A' este 0, 'C' este 2 și 'T' este 19, mesajul este vectorul:



Deci vectorul criptat este dat de:



care corespunde criptotextului 'POH'.

***#COD***

# -\*- coding: utf-8 -\*-

"""

Created on Tue Dec 28 15:59:08 2021

@author: Pavel

"""

keyMatrix = [[0] \* 3 for i in range(3)]

messageVector = [[0] for i in range(3)]

cipherMatrix = [[0] for i in range(3)]

def getKeyMatrix(key):

k = 0

for i in range(3):

for j in range(3):

keyMatrix[i][j] = ord(key[k]) % 65

k += 1

def encrypt(messageVector):

for i in range(3):

for j in range(1):

cipherMatrix[i][j] = 0

for x in range(3):

cipherMatrix[i][j] += (keyMatrix[i][x] \*

messageVector[x][j])

cipherMatrix[i][j] = cipherMatrix[i][j] % 26

def HillCipher(message, key):

getKeyMatrix(key)

for i in range(3):

messageVector[i][0] = ord(message[i]) % 65

encrypt(messageVector)

CipherText = []

for i in range(3):

CipherText.append(chr(cipherMatrix[i][0] + 65))

print("Ciphertext: ", "".join(CipherText))

def main():

message = "ACT"

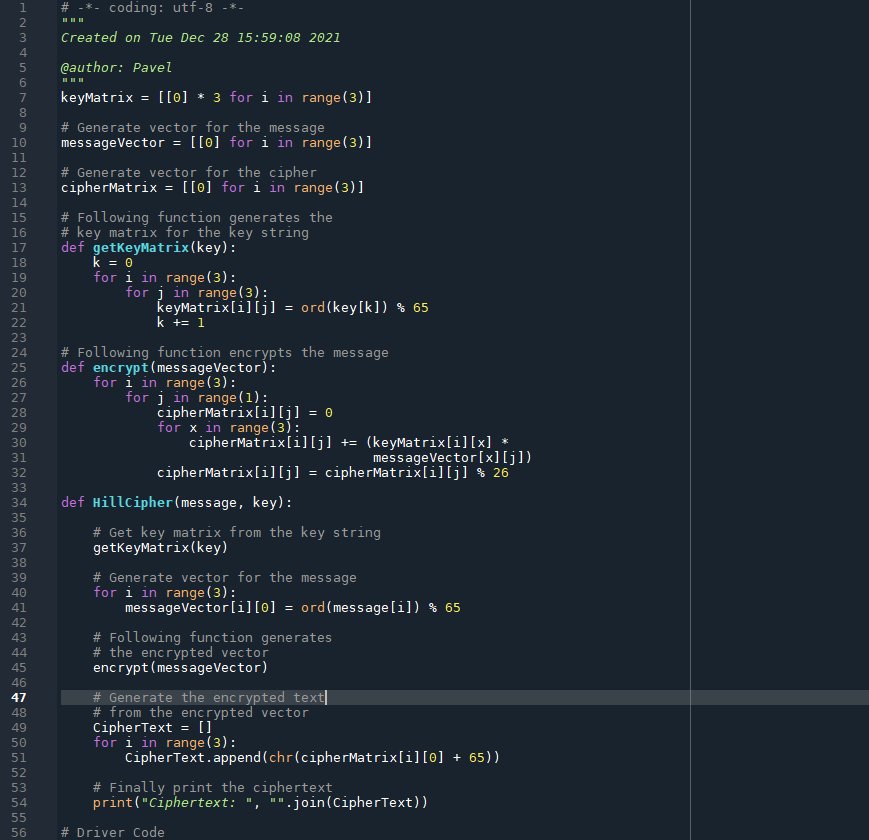
key = "GYBNQKURP"

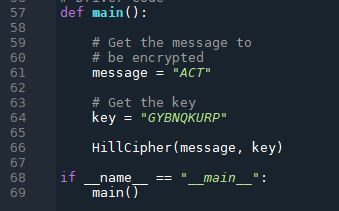
HillCipher(message, key)

if \_\_name\_\_ == "\_\_main\_\_":

main()

***#Screenshots***







***#Program Universal***

# -\*- coding: utf-8 -\*-

"""

Created on Tue Dec 28 16:34:26 2021

@author: Pavel

"""

import numpy as np

from egcd import egcd # pip install egcd

alphabet = "abcdefghijklmnopqrstuvwxyz "

letter\_to\_index = dict(zip(alphabet, range(len(alphabet))))

index\_to\_letter = dict(zip(range(len(alphabet)), alphabet))

def matrix\_mod\_inv(matrix, modulus):

"""Găsim modulul matricei invers cu

Pasul 1) Găsiți determinantul

Pasul 2) Găsiți valoarea determinantului într-un anumit modul (de obicei lungimea alfabetului)

Pasul 3) Luați acel det\_inv ori det\*matricea inversată (aceasta va fi apoi adjunctul) în mod 26

"""

det = int(np.round(np.linalg.det(matrix))) # Pas 1)

det\_inv = egcd(det, modulus)[1] % modulus # Pas 2)

matrix\_modulus\_inv = (

det\_inv \* np.round(det \* np.linalg.inv(matrix)).astype(int) % modulus

) # Pas 3)

return matrix\_modulus\_inv

def encrypt(message, K):

encrypted = ""

message\_in\_numbers = []

for letter in message:

message\_in\_numbers.append(letter\_to\_index[letter])

split\_P = [

message\_in\_numbers[i : i + int(K.shape[0])]

for i in range(0, len(message\_in\_numbers), int(K.shape[0]))

]

for P in split\_P:

P = np.transpose(np.asarray(P))[:, np.newaxis]

while P.shape[0] != K.shape[0]:

P = np.append(P, letter\_to\_index[' '])[:, np.newaxis]

numbers = np.dot(K, P) % len(alphabet)

n = numbers.shape[0] # lungimea mesajului criptat (în cifre)

# Harta înapoi pentru a obține text criptat

for idx in range(n):

number = int(numbers[idx, 0])

encrypted += index\_to\_letter[number]

return encrypted

def decrypt(cipher, Kinv):

decrypted = ""

cipher\_in\_numbers = []

for letter in cipher:

cipher\_in\_numbers.append(letter\_to\_index[letter])

split\_C = [

cipher\_in\_numbers[i : i + int(Kinv.shape[0])]

for i in range(0, len(cipher\_in\_numbers), int(Kinv.shape[0]))

]

for C in split\_C:

C = np.transpose(np.asarray(C))[:, np.newaxis]

numbers = np.dot(Kinv, C) % len(alphabet)

n = numbers.shape[0]

for idx in range(n):

number = int(numbers[idx, 0])

decrypted += index\_to\_letter[number]

return decrypted

def main():

message = input("Introduceti mesajul: ").lower()

K = np.matrix([[3,10,20],[20,19,17], [23,78,17]]) # for length of alphabet = 27

Kinv = matrix\_mod\_inv(K, len(alphabet))

encrypted\_message = encrypt(message, K)

decrypted\_message = decrypt(encrypted\_message, Kinv)

print("Mesaj Clar: " + message)

print("Mesaj Criptat: " + encrypted\_message)

print("Mesaj Decriptat: " + decrypted\_message)

main()

***#Screenshots***

