INDEX

Experiment no: 01

Experiment name: Explain and Implementation of ceasar cipher.

Theory:

ceasar cipher technique:

The ceasar cipher is the simplest and oldest method of cryptography. The ceaser cipher method is based on a mona-alphabetic cipher. and is also called a shift or cipher or additive cipher. Julies ceasar used the shift cipher technique to communicate with his officers. For this reason, the shift cipher technique called the ceasar cipher. The ceasar cipher is a kind of replacement cipher where all letter of plain text is finished.

let's take an example to understand this, suppose we are shifting with 1, then A will be replaced with B, C will be replaced by D and D will be

replaced by C and the process continuous until the entire plan text is finished.

Ceaser cipher is a weak method of cryptography. It can be easily hacked. It means the message encrypted by the method can be easily decrypted.

Plaintext: It is simple message written by the user.

Ciphertext: It is an encrypted message after applying some technique.

The formula of encryption $En(x) = (x+n) \bmod 26$

The formula of decryption $Dn(x) = (x;-n) \bmod 26$

If any case (Dn) value becomes negative in this case, we will add 26 in the negative value.

Where,

E = denotes the encryption
D = denotes the Decryption
x = denotes the letters value
n = denotes the key values (shift value).

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 O=14 P=15 Q=16 R=17 S=18 T=19
U=20 V=21 W=22 X=23 Y=24 Z=25

Example: Use the caesar cipher to encrypt and decrypt the message 'HELLO' and the shift value of the message 15.

Encryption:
We apply encryption formulas by character based on alphabetic order.
The formula of encryption is: $En(x) = (x+n) \bmod 26$.

plaintext	Encryption	ciphertext
H – 7	(7+15) mod 26	22 → W
E – 4	(4+15) mod 26	19 → T
L – 11	(11+15) mod 26	0 → A
L – 11	(11+15) mod 26	0 → A
O – 14	(14+15) mod 26	3 → D

The encrypted message of the plain text 'WTAAD'

Decryption:
We apply decryption formula by character based on alphabetical order.
The formula of decryption is - $Dn(x) = (x-n) \bmod 26$

ciphertext	Decryption	plaintext
W → 22	(22-15) mod 26	7 → H
T → 19	(19-15) mod 26	4 → E
A → 0	(0-15) mod 26	11 → L
A → 0	(0-15) mod 26	11 → L
D → 3	(3-15) mod 26	14 → O

The decrypted message is 'HELLO'.

Source code:

```
impost sys
while (1):
  def encryp (text, k):
  result = " "
  for i in range (len(text)):
  character = text [i]
  if (character. isupper ()):
    if (ord (character) =32):
       result + = "  "
    else:
    result += "  "
    else
    result + = chr ((ord (character) + k -65) % 26
    +65)
  else
  if (ord(character) = = 32):
  result + = "  "
  else:
result + = chr((ord (character)+k-97) % 26+97)
return result

for decryp (text, k):
result = "
for i in range (len (text)):
character = text [i]
if (character. isupper ()):
```

```
if (ord (character) == 32):
result += " "
else:
 result += "
else:
result += chr ((ord (character) - k - 97) % 26 + 97)
return result

choice = ['1', '2', '3']

user choice = input ('Enter 1 for encryption
      2 for Decryption, 3 for exit:")

if user choice == '3']
 sys. exit ()
if user_choice = = '1';
plain_text = input ("Enter your plain text")
key = int(input ("enter key"))
output_cipher_text = encryp (plain_text, key)
print ('cipher text: ' + output_cipher_text)

 if user_choice = = "2";
 cipher_text = input ("Enter your cipher text:')
 key = int (input ("Enter key"))
 print ("original text:" + decryp (cipher_text, 2
```

Output:

Enter a message to encrypt: pustiet
Enter key: reset
Encrypted message: pustict
Decrypted message: pustiet

Experiment No: 02

Experiment Name: Implemantation of mono-alphabetic cipher.

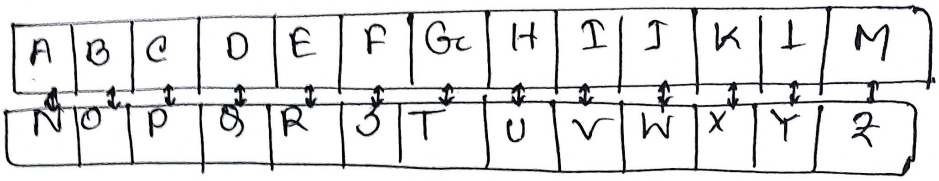
Theory:

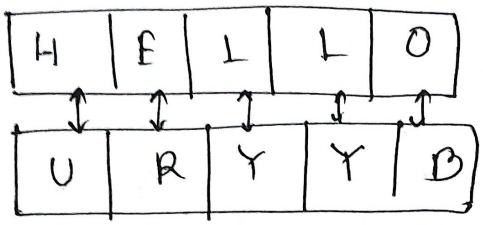
A monoalphabetic cipher is any cipher in which the letters of the plaintext are mapped to ciphertext letters based on a single alphabet key. Substitutions ciphers work by replacing each letter of the plaintext with another letter. For this reason, a mono-alphabetic cipher is also called a simple substitution cipher.

It relies on a fixed replacement structure, meaning the substitution is fixed for each letter of the alphabet. Thus if the latter 'a' is encoded as letter Q then every time 'a' appears in the plantext, it's replaced with the letter Q.

There are many monoalphabetic substitution ciphers, actually infinity many as each latter can be encrypted to any symbol.

not just another letter.





Algorithm:

Encrypt:

```
encrypt (p, key)
    encrypted = " "
for i in p
encrypted . appened (key [i])
    return encrypted
```

Decrypt:

```
decrypt (E, key)
    decrypted = " "
for i in E
decrypted . append (key^-1 [i])
return decrypted.
```

Source code:

```
keys = [
 'q', 'w', 'e', 'r', 't', 'y', 'u', 'i', 'o',
 'p', 'a', 's', 'd', 'f', 'g', 'h', 'j'
 'k', 'l', 'z', 'x', 'c', 'v', 'b', 'n', 'm'
]

plainText = input ('Enter message')
cipherText = ""
for i in plainText:
  cipherText = cipherText + keys[(int (ord (i))-
 97 ) % 26]

print ("Encoded message is:" + cipherText)
decodeText = ""
for i in cipherText:
decode text = decodetext + chr (int (keys, index (i))
+97)
print ('Decode text is:" + decodeText)
```

Output:

a → m
b → b
c → j
d → y
e → a
f → 1
g → 2
h → v
i → e
j → p
k → s
l → f
m → o
n → n
o → x
p → q
q → g
R → d
S → u
t → r
u → i
v → t
w → w
x → k
y → c
z → l

The encrypted text — qiuret a
the decrypted text — pustice

Experiment No: 03

Experiment Name: Implementation of playfair cipher.

Theory:

In this schema, pairs of letters are encrypted instead of single letters as in the case of simple substitution ciphers.

In playfair cipher initially a key table is created. the key table is a 5x5 grid of alphabets. that acts as the key for encrypting the plaintext. Each of the 25 alphabets must be unique and one letter of the alphabet is omitted from the table as we need only 25 alphabets instead of 26. If the plaintext contains J then it is replaced by I.

The sender and the receiver decided on a particular key say 'tutorials'. In a key table, the first character in the table is the phrase, excluding the duplicate letters. the rest of the table will be

filled with the remaining letters of the alphabet in natural order. The key table works to be,

T	U	O	R	I
A	L	S	B	C
D	E	F	G	H
K	M	N	P	Q
V	W	X	Y	Z

Process of playfair cipher:

First, a plaintext message a split into pairs of two letters. If there is an odd number of letters. a Z is added to the last letter. Let us say we want to encrypt the message hide money. It will written as —

HI DE MO NE YZ

The rules of encryption are —

If both the letters are in the same column, take the letter below each one.

T	U	O	R	I
A	L	S	B	C
D	E	F	G	H
K	M	N	P	Q
V	W	X	Y	Z

'H' and 'I' are in same column, hence take letters below them to replace. HI → QC

⇒ If both letters are in the same row, take the letter to the right of each one.

T	U	O	R	I
A	L	S	B	C
D	E	F	G	H
K	M	N	P	Q
V	W	X	Y	Z

'D' and 'E' are in same row, hence take letter to the right of them to replace DE → EF

If neither of the preceding two rules are true, from a rectangle with the two letters and take the letters on the horizontal opposite corners of the rectangle.

T	U	O	R	I
A	L	S	B	C
D	E	F	G	H
K	M	N	P	Q
V	W	X	Y	Z

('M' and 'O' not on same column or on same row. hence from rectangle as shown and replace letter by picking up composite corner letter on same row MO → NU.

Using these rules, the result of the encryption of 'hide money' with the key of tutorials would be —

QE EF NU FM MF ZV

Decrypting the playfair cipher is as simple as doing the same process in reverse. Receiver has the same key and can create the same key table, and then decrypt any message made using that key.

Experiment No: 04

Experiment Name: Implementation of Railfence cipher.

Theory:
The railfence cipher is a transposition cipher that jumbles up the order of the letters of a message using a basic algorithm

The railface cipher works by writing your message an alternate lines across the page and then reading off each line in turn.

For example, let's consider the plaintext.
This is a secret message.

Plaintext THISISASECRETMESSAGE

To encode the message we first write over two lines (the rails of the fence) as follows:

Rail Fence Encoding

T		I		T		A		E		R	
	H		S		S		S		C		E
			T		E		S		G		
				M		S		A		E	

Note that, all white spaces have been removed from the plaintext.

the ciphertext is then read off by by writting the top row first, followed by the bottom row:

ciphertext: TTAERTE SG A SSS CEMS AE

Source code:

```
def removeSpaces(text):
 newText = ""
 for i in text:
     if i == " ":

 else
    newText = newText + i
 return newText
def encryptRailFence(text, key):
rail = [['\n' for i in range(len(text))] for
I in range(key)]
dir_down = False
row, col = 0, 0
for i in range(len(text)):
if (row == 0) or (row == key - 1):
dir_down = not dir_down
rail[row][col] = text[i]
 col += 1
if dir_down:
row += 1
else
 row -= 1
 result = []
 for i in range(key):
 for I in range(len(text)):
  if rail[i][I] != '\n':
 result.append(rail[i][I]).
```

```
return ("", join (result))
def decryptRailFence (cipher, key):
rail = [['/n' for i in range (len (cipher))]
for j in range (key)]
dir_down = None
row, col = 0,0

for i in range (len (cipher)):
if row == 0:
  dir down = True
if row == key -1:
 dir_down = False
rail [row] [col] = '*'
col += 1
if dir_down:
  row -= 1

index = 0
for i in range (key):
for j in range (len(cipher)):
if ((rail [i][j] == '*') and (index < len(cipher))
rail [i] [j] = cipher [index]
 index += 1
result = [ ]
row, col = 0,0
for i in range (len(cipher) ):
if row == 0 :
    dir - down = True
if row  = -key - 1:
        dir-down : False
```

```
if (rail [row] [col] != '*'):
result . append (rail [row ] [col])
col += 1

if dir_down:
    row += 1

else:
  row -= 1
return ("", join (result))

if _name_ == "_main_":
text = input ("Enter the plain text:").
key = int (input ("Enter the Rail fence of
                  depth:))
plain_text = removespaces (tolowercase (text))
print ("\n plain Text: " + plain_text)
print ("cipher text: " + encryptrail Fence
          (plain_text, key))

cipher_text = input ("\n Enter cipher Text:")
key of_depth = int (input ("Enter the depth:"))
print ("original text: " + decryptRail Fence
  (cipher_text , key_of_depth)).
```

Experiment No: 05

Experiment Name: Implementation of vernam cipher.

Theory:

Vernam cipher is a method of encrypting alphabetic text. It is one of the substitution techniques for converting plain text into cipher text. In this mechanism we assign a number to each character of the plain text. like (a=0, b=1, c=2 ... z=25)

Method to tak key: In the vernam cipher algorithm, we take a key to encrypt the plaintext whose length should be equal to the length of the plain text

Encryption Algorithm:

1. Assign a number to each character of the plain-text and the key according to alphabetic order.

Bitwise XOR both the number (corresponding plain text character number and key character number).

3. Subtract the number from 26 if the resulting number is greater than or equal to 26. if it isn't then leave it.

Example : Plaintext - OAK
key - SON

O ==> 14 = 0 1 1 1 0

S ==> 18 = 1 0 0 1 0

Bitwise XOR result : 11100 = 28

Since the resulting number is greater than 26, subtract 26 from it. Then convert then cipher-text character number to the cipher text character.

28-26 = 2 ==> c

cipher text : C

Similarly do the same for the other corresponding character

PT: O A K

NO: 14 00 10

key: 30 N

NO: 18 14 13

New ciphertext is after getting the corresponding character from the resulting number.

CT:- NO: 02 14 07

CT - C O H

Source code:

```
import string
import random

def getMode():
  while True:
  mode = str(input("Encrypt for e, decrypt
  for d and exit for quit")).lower()
  if mode in 'encrypt e decrypt d'.split():
  return mode
  elif mode in "exit quit".split():
  break
  else:
  print("Enter either 'encrypt' or 'e'
  for encryption or 'decrypt' or 'd' for
  Dycryption.")

def get message():

message = str(input("Enter the message:"))
message = message.lower()
return message
def generate key(message):
key = ''.join([random.choice(string.
  ascii_letters) for n in range(len(message))]

key = key.lower()
return key
```

```
def ciphertext(message, key):
encrypted_message = ''
index = 0
for symbol in message:
temp = (alphabet_to_number_dict[symbol] +
alphabet_to_number_dict[key[index]]) %26

while True:
mode = str(input('Encrypt for e, decrypt
for d, exit for quit')).lower()
if mode in 'encrypt e decrypt d'.split():
if mode[0] == 'e':
message = getmessage().
key = generate_key(message)
print('Encryption key: {0}'.formed(key)
elif mode[0] = 'd':
message = getmessage()
key = getkey()
print('ciphertext message: {0}'.format(message))
break
```

output:

Enter plain text : PUST
Enter key : best

77707777 00000001
77707770
77770700 00000700
77770000
77770070 00070070
77700000
77770077 00070077
77700000

The cipher text : OQAA

77701110 00000001
11101111
1111 0000 00000010
11110100
11100000 00010012
11110010
11100000 00010011
11110011

The plaintext : PUST

Experiment No: 06

Experiment Name: Implemantation of poly alphabetic cipher.

Theory:

The alphabetic ey cipher is another nam vigenere cipher. The vigenere cipher is algorithm that is used to encrypting a decrypting the text. The vigenere cipher i an algorithm of encrypting an alphabetic te that uses a series of interwoven ceasen ciphers. It is based on a keyword's letter It is an example of a polyalphabetic substitution cipher. This algorithm is easy to understand and implement. This algo- rithm was first described in 1553 by Giovan Battista Bellaso. It uses a vigener- table o vigenare square for encryption and decryption of the text. the vigenare tabu is also called the tabula recta.

Experiment No: 06

Experiment Name: Implemantation of poly-alphabetic cipher.

Theory:

The alphabetic ey cipher is another name vigenere cipher. The vigenere cipher is an algorithm that is used to encrypting and decrypting the text. The vigenere cipher is an algorithm of encrypting an alphabetic text that uses a series of interwoven ceaser ciphers. It is based on a keyword's letter. It is an example of a polyalphabetic substitution cipher. This algorithm is easy to understand and implement. This algorithm was first described in 1553 by Giovan Battista Bellaso. It uses a vigener table o vigenare square for encryption and decryption of the text the vigenare table is also called the tabula recta

Two methods perform the vigenere cipher—

Method 1: When the vigenere table is given, the encryption and decryption are done using the vigenere table (26 * 26) in this method.

Example:

The plaintext is 'javapoint' and the key is 'BEST'.

to generate a new key then the given key is represented in a circular manner as long as the length of the plain text doesnot equal to the new key

J	A	V	A	P	O	I	N	T
B	E	S	T	B	E	S	T	B

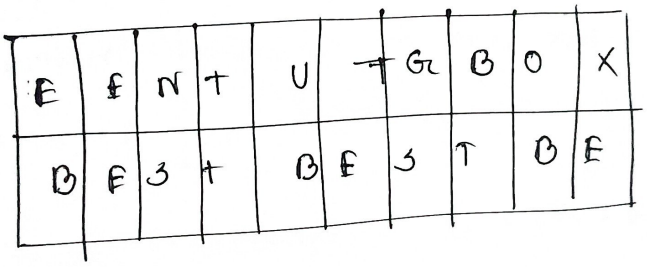
Encryption:

The first letter of the plaintext is combined with the first letter of the key. The column of plain text 'J' and row of key "B". intersects the alphabet of 'K' in the vigenere table, so the first letter of ciphertext is 'K'.

ciphertext = KENTUIGBOX.

Decryption:

Decryption is done by the row of key in the vigenere table. First select the row of the key letter. find the ciphertext letters position in that row, and then select the column the column label of the corresponding ciphertext as the plaintext.



For example in the row of the key, is 'B' and the cipher text is 'K' and this ciphertext letter appears in the column 'J'. that means the first plaintext letters 'J'

Next in the row of the key is 'E' and the cipher text is 'E'. and this ciphertext letter appears in the column 'A' that means the second plaintext letter A.

The process continuous continuously until the ciphertext is finished

plaintext = javapoINT.

Method 2:

When the vigenere table is not given the encryption and decryption are done by vigenere formula in the method (convert the letters (A, J) into the number.

Formula of encryption – $E_i = (P_i + K_i) \bmod 26$

Formula of decryption - $D_i = (E_i - K_i) \bmod 26$

If any case, (D_i) value becomes negative, we will add 26 in the negative value.

E = the encryption

D = decryption

P = plaintest

k = key

Source code

```
def generatekey(string, key):
key = list(key)
if len(string) == len(key):
return(key)

else:
for i in range(len(string)
                len(key)):
    key.append(key[i % len(key)])
return("".join(key))

def ciphertext(string, key):
ciphertext = []
for i in range(len(string)):
x = (ord(string[i]) + ord(key[i])) %26
x += ord('A')
cipher_text.append(chr(x))
return("".join(cipher_text))

def originalText(ciphertext, key):
orig_text = []
for i in range(len(ciphertext)):
x = (ord(ciphertext[i]) - ord(key[i]) +26)%26
x += ord('A')
orig_text.append(chr(x))
return("".join(orig_text))
```

```
if name == '__main__':
string = input ('Enter plain Text:')
keyword = input ('Enter the key:')
print ("\n plain Text : " +string)
key = generatkey (string, keyword)
print ("keyword : " + key)
ciphere_text = ciphertext (string key)
print ('cipher text :', ciphertext)
print ('original / Decrypted Text :'
originalText (ciphere_text))
```

output:

Enter the plaintext : pusheen

Enter the keyword : bestice

ciphertext : QYKMQET

plaintext : PUSTPOE

Experiment No: 07

Experiment Name: Implementation of hill cipher

Theory:

The hill cypher is a polygraphic substitution cipher based on linear Algebra. It was invented by lesters. Hill in the year 1929. In simple words it is a cryptography algorithm used to encrypt and decrypt data for the purpose of data security.

The algorithm uses matrix calculations used in linear algebra. If the users to understand if we have the basic knowledge of matrix multiplication, module calculation and the inverse calculation of matrices.

In hill cipher algorithm every letter (A-Z) is represented by a number module 26. usually the simple substitution scheme is used where A=0 B=1 C=2 ... Z=25 in order to use 2x2 key matrix.

Encryption:

To encrypt the text using hill cipher we need to perform the following equation-

$$E(k, p) = (k * p) \bmod 26$$

k is key matrix, p is plaintext in vector form. Matrix multiplication of k and p generates the encrypted ciphertext.

Steps for encryption:

Step 1: let's say our key text (2×2) is DCDF. convert this key using a substitution schema into a 2×2 key matrix as shown-

DCDF → $\begin{bmatrix} D & D \\ C & F \end{bmatrix}$ → $\begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix}$

Step 2- Now we will convert our plain-text into vector form. Since the key matrix is 2×2. the vector must be 2×1 for matrix multiplication

In our case, plain text is TEXT that is four letters long word. thus we can put in a 2x1 vector and then substitute as.

$$TEXT \rightarrow \begin{bmatrix} T \\ E \end{bmatrix} \begin{bmatrix} X \\ T \end{bmatrix} \rightarrow \begin{bmatrix} 19 \\ 4 \end{bmatrix} \begin{bmatrix} 23 \\ 19 \end{bmatrix}$$

Step 3: Multiply the key matrix with each 2x1 plaintext vector and take the module of result (2x1) vectors by 26. then concatenate the results and we get the encrypted or ciphertext as RGWL

$$\begin{bmatrix} D & D \\ E & F \end{bmatrix} \times \begin{bmatrix} T \\ E \end{bmatrix} \rightarrow \begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix} \begin{bmatrix} 19 \\ 4 \end{bmatrix} = \begin{bmatrix} 69 \\ 58 \end{bmatrix} \% 26.$$

$$\begin{bmatrix} 17 \\ 6 \end{bmatrix} \rightarrow \begin{bmatrix} R \\ G \end{bmatrix}$$

$$\begin{bmatrix} D & D \\ C & F \end{bmatrix} \times \begin{bmatrix} X \\ T \end{bmatrix} \rightarrow \begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix} \begin{bmatrix} 2 & 3 \\ 1 & 9 \end{bmatrix} \cdot \begin{bmatrix} 726 \\ 741 \end{bmatrix} \%$$

$$26 \cdot \begin{bmatrix} 22 \\ 11 \end{bmatrix} = \begin{bmatrix} W \\ L \end{bmatrix}$$

RGWL

Decryption:

To encrypt the text using hill cipher, we need to follow the following operation

$$D(k.e) = (k-1 \times e) \bmod 26$$

where k is the key matrix and C is the ciphertext in vector form. Matrix multiplication of inverse of key matrix k and ciphertext e generates the decrypted plaintext

Step 1: Calculate the inverse of the key matrix. First we need to find the determinant of the key matrix. Here the extended Euclidean algorithm is used to get modulo multiplicative inverse of key matrix determinat.

$$k^{-1} \% 26 = \begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix} \% 26$$

$$= \left((3\times 2) - (3\times 2)\right)^{-1} \times \begin{bmatrix} 5 & -3 \\ -2 & 3 \end{bmatrix} \% 26 =$$

$$3 \begin{bmatrix} 5 & 23 \\ 24 & 3 \end{bmatrix}$$

$$= \begin{bmatrix} 15 & 69 \\ 72 & 9 \end{bmatrix} \% \cdot \begin{bmatrix} 15 & 17 \\ 20 & 9 \end{bmatrix}$$

Step 2: Now we multiply the 2×1 blocks of ciphertext and the inverse of the key matrix. The resultant block after concatenation is the plain text that we have encrypted i.e. TEXT

$$\begin{bmatrix} 15 & 17 \\ 20 & 9 \end{bmatrix} \times \begin{bmatrix} 17 \\ 6 \end{bmatrix} = \begin{bmatrix} 357 \\ 394 \end{bmatrix} \% 26 =$$

$$\begin{bmatrix} 19 \\ 4 \end{bmatrix} \rightarrow \begin{bmatrix} T \\ E \end{bmatrix}$$

$$\begin{bmatrix} 15 & 17 \\ 20 & 9 \end{bmatrix} \times \begin{bmatrix} 22 \\ 17 \end{bmatrix} = \begin{bmatrix} 517 \\ 539 \end{bmatrix} \% 26 = \begin{bmatrix} 23 \\ 19 \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} X \\ T \end{bmatrix}$$

Source code:

```
def getkeyMatrix (key, n):
    k=0
for i in range (n):
    for j in range (n):
        keymatrix [i] [j] = ord (key [k]) %65
        k+=1

def encrypt (message vector, n):
    for i in range (n):
        for j in range (1):
        ciphermatrix [i] [j] = 0
    for x in range (n):
        cipher matrix [i] [j] = cipher matrix
                                [i] [j] % 26

def HillCipher (message, key, n):
getkeymatrix (key, n)
for i in range (n):
message vector [i] [0] = ord (message [i]) %65
encrypt (message vector, n)
cipher text = [ ]
for i in range (n):
cipher text append (chr (ciphermatrix [i] [0]+
65))
```

```
Print ("ciphertext :", ''.join(ciphertext))

message = "DC"
key = 'HIGrF'
n = len(message)
keymatrix = [[0] * n for i in range (n)]
print (keymatrix)
messageVector = [[0] for i in range(n)]
cipherMatrix = [[0] for i in range (n)]
HillCipher (message, key, n)
```

Output:

Enter the key matrix 3 3
9 5

Enter plaintext jeepust
plaintext : jeepust
Encryption text: EAFFKAFM
Determinant value: 9

Inverse matrix:
5 -3
-2 3

Modify matrix:
15 17
20 9

Encryption text: EAFFKAFM
Decryption text: JEEPUSTA