

**B. Tech – CE | Semester: VI | Subject: NIS**  
**Lab 3**

**Vigenere Cipher :** The vigenere cipher is an algorithm of encrypting alphabetic text that uses a series of Caesar ciphers. It is based on the keyword's letter. It is an example of a polyalphabetic substitution cipher, Because for the same alphabet it has a different encryption key and encrypted character differ and it has the same encryption for different plain text characters also. This algorithm is easy to understand and implement.

Here encryption key is → BOP

Plain text : P R O S P E R

ASCII : 15 17 14 18 15 4 17

Key : B O P B O P B

Cipher : Q F D T T D S

Key : 16 5 3 19 3 19 18

In encryption two same “P” in plain text encrypted as a “Q” and “T”, while two different character “O” and “E” of plain text is encrypted same as a “D”

**Method 1 :**

vigenere algebraically formula. In this method the first thing to do is convert all alphabet a-z into numbers 0-25.

→ The encryption formula is :-  $E(i) = [ P(i) + K(i) ] \bmod 26$  -

→ The decryption formula is :  $D(i) = [ E(i) - K(i) ] \bmod 26$

→ In the case of decryption whenever d(i) value becomes negative then we will add 26 into negative value.

→ in the following code I have used this method to encrypt and decrypt the text and implemented a vigenere cipher.

## Method 2 :

vigenere table → When the vigenere table given the encryption and decryption are done using the vigenere table (26 \* 26 matrix).

**Task 1 : Write a Program to do Encryption and Decryption using Vigenere Cipher.**

**Source Code : Python**

```
#vigenere cipher
from collections import defaultdict

d = defaultdict(lambda: " ")
for i in range(27):
    d[chr(97+i)] = i

def get_key(val):
    for key, value in d.items():
        if val == value:
            return key
    return " "

def Encryption(plain_text, key):
    temp = list()
    cipher_text = ""
    count = 0
    print()
    for i in plain_text:
        if i == " ":
            count -= 1
            temp.append(" ")
        else:
            temp.append((d[i] + d[key[count % len(key)]]) % 26)
            count += 1

    # print(temp)
    for i in temp:
        cipher_text += get_key(i)
    print("cipher text is ->", cipher_text)
    return cipher_text
```

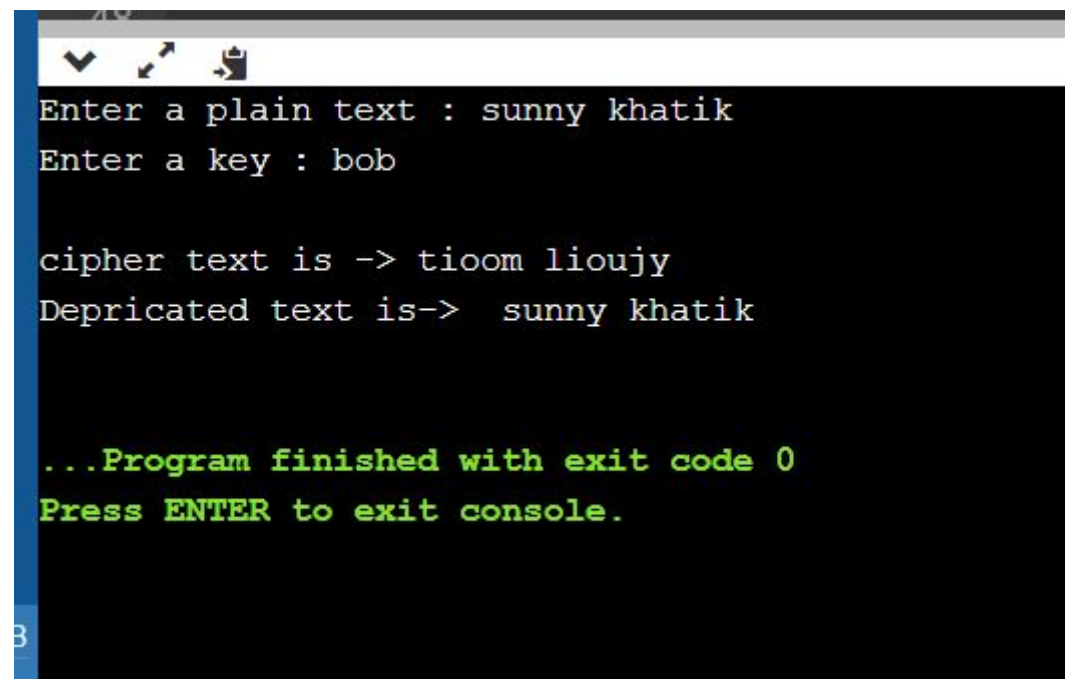
```

def Decryption( cipher_text , key):
    decripted_text = ""
    count = 0
    temp = list()
    for i in cipher_text:
        if i == " ":
            count-=1
            temp.append(" ")
        else:
            temp.append((d/i)-d/key/count%(len(key))//)%26)
            count+=1
    for i in temp:
        decripted_text+=get_key(i)
    print("Deprecated text is-> ", decripted_text)
    return decripted_text

if __name__ == "__main__":
    plain_text = input("Enter a plain text : ")
    key = input("Enter a key : ")
    cipher_text = Encryption( plain_text , list(key))
    decripted_text = Decryption( cipher_text , key)

```

Output :



```

Enter a plain text : sunny khatik
Enter a key : bob

cipher text is -> tioom lioujy
Deprecated text is-> sunny khatik

...Program finished with exit code 0
Press ENTER to exit console.

```

## Cryptanalysis of vigenere cipher :

→ Cryptanalysis of the Vigenere cipher has 2 main steps: identify the period of the cipher (the length of the key), then find the specific key. To identify the period we use a test based on the index of coincidence, to find the specific key we use the chi-squared test. - Chi-squared test is very useful in machine learning and it is used for comparing distribution of probabilities.

**Task 2 : Do the Cryptanalysis of Vigenere Cipher (Use sufficiently large CipherText). Use Index of Coincidence to verify the guessed Key Length. Use Mutual Index of Coincidence to guess the Key.**

Source Code : Python

```
from string import uppercase
from operator import itemgetter

def vigenere_decrypt(target_freqs, input):
    nchars = len(uppercase)
    ordA = ord('A')
    sorted_targets = sorted(target_freqs)

    def frequency(input):
        result = [c, 0.0] for c in uppercase
        for c in input:
            result[c - ordA][1] += 1
        return result

    def correlation(input):
        result = 0.0
        freq = frequency(input)
        freq.sort(key=itemgetter(1))

        for i, f in enumerate(freq):
```

```

        result += f/1/ * sorted_targets/i/
    return result

cleaned = [ord(c) for c in input.upper() if c.isupper()]
best_len = 0
best_corr = -100.0

# Assume that if there are less than 20 characters
# per column, the key's too long to guess
for i in xrange(2, len(cleaned) // 20):
    pieces = [[] for _ in xrange(i)]
    for j, c in enumerate(cleaned):
        pieces[j % i].append(c)

    # The correlation seems to increase for smaller
    # pieces/longer keys, so weigh against them a little
    corr = -0.5 * i + sum(correlation(p) for p in pieces)

    if corr > best_corr:
        best_len = i
        best_corr = corr

if best_len == 0:
    return ("Text is too short to analyze", "")

pieces = [[] for _ in xrange(best_len)]
for i, c in enumerate(cleaned):
    pieces[i % best_len].append(c)

freqs = [frequency(p) for p in pieces]

key = ""
for fr in freqs:
    fr.sort(key=itemgetter(1), reverse=True)

    m = 0
    max_corr = 0.0
    for j in xrange(nchars):
        corr = 0.0
        c = ordA + j
        for frc in fr:
            d = (ord(frc/0)) - c + nchars % nchars

```

```

        corr += freq/1/ * target_freqs/d/

    if corr > max_corr:
        m = j
        max_corr = corr

    key += chr(m + ordA)

r = (chr((c - ord(key/i % best_len)) + nchars) % nchars + ordA)
    for i, c in enumerate(cleaned))
return (key, "".join(r))

def main():
    encoded = ""

    vptnvffuntsharptymjwzirappljmhhqvsbwlzzygvtyitarptyiougxiuydtgzhhvmmum
shwkzgstfmekvmpkswdgbilvjlmglmjfqwioiivknulvvfemioiemojtywdsajtwmtcgluy
sdsumfbieugmvalvckjduetukatymvkvqzhvqvgvptytjwwldyeevquhlulwpkt""

    english_frequencies = /
    0.08167, 0.01492, 0.02782, 0.04253, 0.12702, 0.02228, 0.02015,
    0.06094, 0.06966, 0.00153, 0.00772, 0.04025, 0.02406, 0.06749,
    0.07507, 0.01929, 0.00095, 0.05987, 0.06327, 0.09056, 0.02758,
    0.00978, 0.02360, 0.00150, 0.01974, 0.00074/

    (key, decoded) = vigenere_decrypt(english_frequencies, encoded)
    print ("Key:", key)
    print ("\nText:", decoded)

main()

```

Output :

```

76 def main():
77     encoded = ""
78     vptnvffuntsharptymjwzirappljmhhqvsbwlzzygvtyitarptyiougxiuydtgzhhvmmum
79 shwkzgstfmekvmpkswdgbilvjlmglmjfqwioiivknulvvfemioiemojtywdsajtwmtcgluy
80 sdsumfbieugmvalvckjduetukatymvkvqzhvqvgvptytjwwldyeevquhlulwpkt""
81
82     english_frequencies = [
83         0.08167, 0.01492, 0.02782, 0.04253, 0.12702, 0.02228, 0.02015,
84         0.06094, 0.06966, 0.00153, 0.00772, 0.04025, 0.02406, 0.06749,
85         0.07507, 0.01929, 0.00095, 0.05987, 0.06327, 0.09056, 0.02758,
86         0.00978, 0.02360, 0.00150, 0.01974, 0.00074]
87
88     (key, decoded) = vigenere_decrypt(english_frequencies, encoded)
89     print "Key:", key
90     print "\nText:", decoded

```

Result

Python main.py

Key: CIPHERS

Text: THEGRONSFELDCIPHERISEXACTLYTHESAMEASTHEVIGENERECIPHEREXCEPTNUMBERSAREUSEDASTHEKEYINSTEADOFLETTERS.THEREISNOOTHERDIFFERENCE.THE NUMBERSMAYBEPICKEDFROMANYSEQUENCEEGTHEFIBONACCISERIESORSOMEOTHERPSEUDORANDOMSEQUENCE