Name: Vishal Indradev Chauhan

PRN: 2020BTECS00090

Class: Final Year - CSE

Assignment No 1

Cryptography and Network Security Lab (5CS453)

Title:

Encryption and Decryption using Ceaser Cipher.

Aim:

To Study and Implement Encryption and Decryption using Ceaser Cipher

Theory:

- Caesar Cipher, also known as the Shift Cipher, is one of the simplest and oldest encryption techniques used to secure information.
- It's a type of substitution cipher where each letter in the plaintext is shifted a certain number of places down or up the alphabet.
- The number of positions a letter is shifted is determined by a key.

Encryption:

In Encryption, input is a Plain text and output is a Cipher text.

Step 1: Choose a secret key (a positive integer).
Step 2: Take the plaintext message you want to encrypt.
Step 3: Shift each letter in the message forward in the alphabet by the key positions.
Step 4: Non-alphabetical characters remain unchanged.
Step 5: The result is the ciphertext, the encrypted message.

Decryption:

In Decryption, input is a Cipher text and output is a Plain text.

Step 1: Have the same key used for encryption.
Step 2: Take the ciphertext (the encrypted message).
Step 3: Shift each letter in the ciphertext backward in the alphabet by the key positions.
Step 4: Non-alphabetical characters remain unchanged.
Step 5: The result is the plaintext, the original message.

Code:

```
Assign1 > both.py
 1   import string
 2
 3   # Function to check if a word is meaningful
 4   def is_valid_word(word, dictionary):
 5       return word in dictionary
 6
 7   def caesar_cipher_encrypt(plaintext, key):
 8       ciphertext = ""
 9       for c in plaintext:
10           if c.isalpha():
11               base = ord('A') if c.isupper() else ord('a')
12               encrypted_char = chr(((ord(c) - base + key) % 26) + base)
13               ciphertext += encrypted_char
14           else:
15               ciphertext += c   # Preserve non-alphabetic characters as is
16       return ciphertext
17
18   def caesar_cipher_decrypt(ciphertext, key):
19       plaintext = ""
20       for c in ciphertext:
21           if c.isalpha():
22               base = ord('A') if c.isupper() else ord('a')
23               decrypted_char = chr(((ord(c) - base - key + 26) % 26) + base)
24               plaintext += decrypted_char
25           else:
26               plaintext += c   # Preserve non-alphabetic characters as is
27       return plaintext
28
29   def print_all_combinations(ciphertext):
30       print("All possible plaintext combinations:")
31       for key in range(26):
32           plaintext = caesar_cipher_decrypt(ciphertext, key)
33           print(f"Key {key}: {plaintext}")
```

```
Assign1 > both.py > {} string
34
35   if __name__ == "__main__":
36       valid_words = ["hello", "world", "example", "brijesh", "is", "a", "good", "boy", "cipher", "decrypt", "meaningful", "pl
37
38       while True:
39           print("Menu:")
40           print("1. Encrypt")
41           print("2. Decrypt and Filter Meaningful Words")
42           print("3. Print All Possible Plaintext Combinations")
43           print("4. Exit")
44           choice = input("Enter your choice (1/2/3/4): ")
45
46           if choice == "1":
47               input_text = input("Enter the plaintext: ")
48               key = int(input("Enter the key (a positive integer): "))
49               ciphertext = caesar_cipher_encrypt(input_text, key)
50               print(f"Ciphertext: {ciphertext}")
51
52           elif choice == "2":
53               ciphertext = input("Enter the ciphertext: ")
54               key = int(input("Enter the key (a positive integer): "))
55               decrypted_text = caesar_cipher_decrypt(ciphertext, key)
56               print(f"Decrypted Text: {decrypted_text}")
57
58               meaningful_text = " ".join(word for word in decrypted_text.split() if is_valid_word(word, valid_words))
59               print(f"Meaningful Text: {meaningful_text}")
60
61           elif choice == "3":
62               ciphertext = input("Enter the ciphertext: ")
63               print_all_combinations(ciphertext)
64
65           elif choice == "4":
66               print("Exiting the program.")
67               break
```

Output:

```
PS C:\Users\ANUSHKA\Documents\BTech\CNS\Assign1> python -u "c:\Users\ANUSHKA\Documents\BTech\CNS\Assign1\both.py"
Menu:
1. Encrypt
2. Decrypt and Filter Meaningful Words
3. Print All Possible Plaintext Combinations
4. Exit
Enter your choice (1/2/3/4): 1
Enter the plaintext: hey there, anushka here!
Enter the key (a positive integer): 5
Ciphertext: mjd ymjwj, fszxmpf mjwj!

Menu:
1. Encrypt
2. Decrypt and Filter Meaningful Words
3. Print All Possible Plaintext Combinations
4. Exit
Enter your choice (1/2/3/4): 2
Enter the ciphertext: mjd ymjwj, fszxmpf mjwj!
Enter the key (a positive integer): 5
Decrypted Text: hey there, anushka here!
```

```
Menu:
1. Encrypt
2. Decrypt and Filter Meaningful Words
3. Print All Possible Plaintext Combinations
4. Exit
Enter your choice (1/2/3/4): 3
Enter the ciphertext: mjd ymjwj, fszxmpf mjwj!
All possible plaintext combinations:
Key 0: mjd ymjwj, fszxmpf mjwj!
Key 1: lic xlivi, erywloe livi!
Key 2: khb wkhuh, dqxvknd khuh!
Key 3: jga vjgtg, cpwujmc jgtg!
Key 4: ifz uifsf, bovtilb ifsf!
Key 5: hey there, anushka here!
Key 6: gdx sgdqd, zmtrgjz gdqd!
Key 7: fcw rfcpc, ylsqfiy fcpc!
Key 8: ebv qebob, xkrpehx ebob!
Key 9: dau pdana, wjqodgw dana!
Key 10: czt oczmz, vipncfv czmz!
Key 11: bys nbyly, uhombeu byly!
Key 12: axr maxkx, tgnladt axkx!
Key 13: zwq lzwjw, sfmkzcs zwjw!
Key 14: yvp kyviv, reljybr yviv!
Key 15: xuo jxuhu, qdkixaq xuhu!
Key 16: wtn iwtgt, pcjhwzp wtgt!
Key 17: vsm hvsfs, obigvyo vsfs!
Key 18: url gurer, nahfuxn urer!
Key 19: tqk ftqdq, mzgetwm tqdq!
Key 20: spj espcp, lyfdsvl spcp!
Key 21: roi drobo, kxecruk robo!
Key 22: qnh cqnan, jwdbqtj qnan!
Key 23: pmg bpmzm, ivcapsi pmzm!
Key 24: olf aolyl, hubzorh olyl!
Key 25: nke znkxk, gtaynqg nkxk!
```

```
Menu:
1. Encrypt
2. Decrypt and Filter Meaningful Words
3. Print All Possible Plaintext Combinations
4. Exit
Enter your choice (1/2/3/4): 4
Exiting the program.
PS C:\Users\ANUSHKA\Documents\BTech\CNS\Assign1>
```

Name: Vishal Indradev Chauhan

PRN: 2020BTECS00090

Class: Final Year - CSE

Assignment No 2

Cryptography and Network Security Lab (5CS453)

Title:

Encryption and Decryption using Transposition Cipher Technique.

Aim:

To Study and Implement Encryption and Decryption using Rail Fence Transposition Cipher Technique and Columnar Transposition Cipher Technique

Theory:

1. **Rail Fence Transposition Cipher Technique**

⬚

The Rail Fence Transposition Cipher, also known as the Zigzag Cipher, is a simple columnar transposition cipher technique.

It involves arranging the plaintext characters in a zigzag pattern across multiple rows, known as "rails," and then reading them off row by row to create the encrypted message.

While this cipher is easy to understand and implement, it lacks strong security and is mainly used for educational purposes or simple puzzles.

Encryption:

- Choose the number of rails (rows) for the zigzag pattern.
- Write the message diagonally across the rails, moving up and down.
- Read the characters row by row to form the encrypted message.

Decryption:

- Create the zigzag pattern with the chosen number of rails.
- Leave blank spaces in the pattern for characters to be placed.
- Fill in the blanks with the encrypted characters, row by row.
- Read the characters diagonally to retrieve the original message.

Advantages:

- Easy to understand and implement.
- Provides basic encryption and breaks up character repetition.

Disadvantages:

- Not secure against modern cryptanalysis.
- Security depends on the number of rails, making it less practical for strong encryption.

CODE:

```
Assign2 > reilFence_P.py > ...
 3    class Solution:
 4        def convert(self, s, n):
 5            if n == 1:
 6                return s
 7
 8            result = []
 9            length = len(s)
10            k = 2 * (n - 1)
11
12            for i in range(n):
13                j = i
14                while j < length:
15                    result.append(s[j])
16                    if i != 0 and i != n - 1:
17                        k1 = k - (2 * i)
18                        k2 = j + k1
19                        if k2 < length:
20                            result.append(s[k2])
21                    j += k
22            return ''.join(result)
23
24        def reverse_convert(self, s, n):
25            if n == 1:
26                return s
27
28            original_string = [''] * len(s)
29            length = len(s)
30            k = 2 * (n - 1)
31            idx = 0
32
33            for i in range(n):
34                j = i
35                while j < length:
36                    original_string[j] = s[idx]
```

```
Assign2 >  reilFence_P.py > ...
 34                    j = 1
 35                while j < length:
 36                    original_string[j] = s[idx]
 37                    idx += 1
 38                    if i != 0 and i != n - 1:
 39                        k1 = k - (2 * i)
 40                        k2 = j + k1
 41                        if k2 < length:
 42                            original_string[k2] = s[idx]
 43                            idx += 1
 44                    j += k
 45
 46            return ''.join(original_string)
 47
 48
 49    # Taking input from the user
 50    input_str = input("Enter the input string: ")
 51    input_n = int(input("Enter the value of n: "))
 52
 53    solution = Solution()
 54    converted_string = solution.convert(input_str, input_n)
 55    print("Converted string:", converted_string)
 56
 57    reverse_input_n = int(input("Enter the value of n for reverse conversion: "))
 58    reverse_plain_text = solution.reverse_convert(converted_string, reverse_input_n)
 59    print("Reverse converted plain text:", reverse_plain_text)
```

OUTPUT:

```
PS C:\Users\ANUSHKA\Documents\BTech\CNS> python -u "c:\Users\ANUSHKA\Documents\BTech\CNS\Assign2\reilFence_P.py"
Enter the input string: hey .. have a great day!
Enter the value of n: 3
Converted string: h.aaede .hv  ra a!y egty
Enter the value of n for reverse conversion: 3
Reverse converted plain text: hey .. have a great day!
PS C:\Users\ANUSHKA\Documents\BTech\CNS> python -u "c:\Users\ANUSHKA\Documents\BTech\CNS\Assign2\reilFence_P.py"
Enter the input string: helloo.. yeahh, wish you the same!
Enter the value of n: 3
Converted string: ho hw   eelo.yah ihyutesm!l.e,soha
Enter the value of n for reverse conversion: 3
Reverse converted plain text: helloo.. yeahh, wish you the same!
PS C:\Users\ANUSHKA\Documents\BTech\CNS>
```

2. Columnar Transposition Cipher Technique

⬚

The Columnar Transposition Cipher is a more advanced transposition cipher technique that involves reordering the characters of a message based on a chosen keyword or key phrase.

It provides a higher level of security compared to simpler ciphers like the Rail Fence Cipher. Here's how the Columnar Transposition Cipher works:

Encryption:

- Choose a keyword or key phrase. The unique characters of the keyword determine the order of columns in the transposition grid.
- Write the message row by row into a grid, using the keyword to determine the order of columns.
- Read the characters column by column to obtain the encrypted message.

Decryption:

- Use the keyword to determine the order of columns in the transposition grid.
- Write the encrypted message into the grid column by column.
- Read the characters row by row to retrieve the original plaintext.

Advantages:

- Offers stronger security compared to simpler ciphers.
- Security depends on the length and uniqueness of the keyword.

Disadvantages:

- Can be vulnerable to attacks if the keyword is short or easily guessed.
- May require additional padding characters for messages that don't fit evenly into the grid.

CODE:

```
Assign2 > columnar.py > encryptMessage
 1    import math
 2
 3    # Encryption
 4    def encryptMessage(msg,key):
 5        cipher = ""
 6
 7        k_indx = 0
 8
 9        msg_len = float(len(msg))
10        msg_lst = list(msg)
11        key_lst = sorted(list(key))
12
13        # calculate column of the matrix
14        col = len(key)
15
16        # calculate maximum row of the matrix
17        row = int(math.ceil(msg_len / col))
18
19        # add the padding character '_' in empty
20        # the empty cell of the matix
21        fill_null = int((row * col) - msg_len)
22        msg_lst.extend('_' * fill_null)
23
24        # create Matrix and insert message and
25        # padding characters row-wise
26        matrix = [msg_lst[i: i + col]
27                    for i in range(0, len(msg_lst), col)]
28
29        # read matrix column-wise using key
30        for _ in range(col):
31            curr_idx = key.index(key_lst[k_indx])
32            cipher += ''.join([row[curr_idx]
33                            for row in matrix])
34            k_indx += 1
```

```
Assign2 > 🐍 columnar.py > ⊘ encryptMessage
35
36          return cipher
37
38      # Decryption
39      def decryptMessage(cipher,key):
40          msg = ""
41
42          # track key indices
43          k_indx = 0
44
45          # track msg indices
46          msg_indx = 0
47          msg_len = float(len(cipher))
48          msg_lst = list(cipher)
49
50          # calculate column of the matrix
51          col = len(key)
52
53          # calculate maximum row of the matrix
54          row = int(math.ceil(msg_len / col))
55
56          # convert key into list and sort
57          # alphabetically so we can access
58          # each character by its alphabetical position.
59          key_lst = sorted(list(key))
60
61          # create an empty matrix to
62          # store deciphered message
63          dec_cipher = []
64          for _ in range(row):
65              dec_cipher += [[None] * col]
```

```
Assign2 >  columnar.py >  decryptMessage
67          for _ in range(col):
68              curr_idx = key.index(key_lst[k_indx])
69
70              for j in range(row):
71                  dec_cipher[j][curr_idx] = msg_lst[msg_indx]
72                  msg_indx += 1
73              k_indx += 1
74
75          # convert decrypted msg matrix into a string
76          try:
77              msg = ''.join(sum(dec_cipher, []))
78          except TypeError:
79              raise TypeError("This program cannot",
80                              "handle repeating words.")
81
82          null_count = msg.count('_')
83
84          if null_count > 0:
85              return msg[: -null_count]
86          return msg
87
88
89      def main():
90          msg = input("Enter the message to encrypt: ")
91          key = input("Enter the encryption key (permutation of numbers, e.g., '3124'): ")
92
93          cipher = encryptMessage(msg, key)
94          print("Encrypted message:", cipher)
95
96          decrypted_message = decryptMessage(cipher, key)
97          print("Decrypted message:", decrypted_message)
98
99      if __name__ == "__main__":
100         main()
```

OUTPUT:

```
PS C:\Users\ANUSHKA\Documents\BTech\CNS> python -u "c:\Users\ANUSHKA\Documents\BTech\CNS\Assign2\columnar.py"
Enter the message to encrypt: heyy there.. how was your day?
Enter the encryption key (permutation of numbers, e.g., '3124'): 4213
Encrypted message: yh.oaod_etehwy ?ye.wsua_h r    ry
Decrypted message: heyy there.. how was your day?
PS C:\Users\ANUSHKA\Documents\BTech\CNS> python -u "c:\Users\ANUSHKA\Documents\BTech\CNS\Assign2\columnar.py"
Enter the message to encrypt: yeahh.. it was a great day:)
Enter the encryption key (permutation of numbers, e.g., '3124'): 4123
Encrypted message: e.tsgtya.  r :h waed)yhia aa
Decrypted message: yeahh.. it was a great day:)
PS C:\Users\ANUSHKA\Documents\BTech\CNS> 
```

Name: Vishal Indradev Chauhan

PRN: 2020BTECS00090

Class: Final Year - CSE

Assignment No 3

Cryptography and Network Security Lab (5CS453)

Title:

Encryption and Decryption using Playfair Cipher Technique.

Aim:

To Study and Implement Encryption and Decryption using Playfair Cipher Technique.

Theory:

Playfair Cipher Technique

⬜

The Playfair Cipher Technique is a substitution cipher that encrypts pairs of characters (digraphs) from the plaintext using a 5x5 key square matrix.

The matrix is constructed from a keyword, with duplicate letters removed and the keyword letters placed at the beginning.

Encryption involves applying rules based on the positions of the letters within the key square.

If the letters are in the same row, column, or form a rectangle, they are replaced by specific neighboring letters.

Encryption:

- Divide the plaintext into digraphs.
- Apply rules based on the positions of letters in the key square to replace each digraph.

Decryption:

- Divide the ciphertext into digraphs.
- Apply the rules in reverse to each digraph to retrieve the original plaintext.

Advantages:

- Enhanced security due to digraphs and key square usage.
- Reduces susceptibility to frequency analysis.
- Key square generation is straightforward using a keyword.

Disadvantages:

- Complexity increases with handling various cases (same row, column, rectangle).
- Security depends on the keyword and arrangement of the key square.

CODE:

```
1    def toLowerCase(text):
2        return text.lower()
3
4    def removeSpaces(text):
5        newText = ""
6        for i in text:
7            if i == " ":
8                continue
9            else:
10               newText = newText + i
11       return newText
12
13   def Diagraph(text):
14       Diagraph = []
15       group = 0
16       for i in range(2, len(text), 2):
17           Diagraph.append(text[group:i])
18           group = i
19       Diagraph.append(text[group:])
20       return Diagraph
21
22   def FillerLetter(text):
23       k = len(text)
24       if k % 2 == 0:
25           for i in range(0, k, 2):
26               if text[i] == text[i+1]:
27                   new_word = text[0:i+1] + str('x') + text[i+1:]
28                   new_word = FillerLetter(new_word)
29                   break
30               else:
31                   new_word = text
```

Assign3 > 🐍 Playfair.py > ⓨ Diagraph

```python
32          else:
33              for i in range(0, k-1, 2):
34                  if text[i] == text[i+1]:
35                      new_word = text[0:i+1] + str('x') + text[i+1:]
36                      new_word = FillerLetter(new_word)
37                      break
38                  else:
39                      new_word = text
40          return new_word
41
42
43  list1 = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'k', 'l', 'm',
44           'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
45
46  # Function to generate the 5x5 key square matrix
47
48
49  def generateKeyTable(word, list1):
50      key_letters = []
51      for i in word:
52          if i not in key_letters:
53              key_letters.append(i)
54
55      compElements = []
56      for i in key_letters:
57          if i not in compElements:
58              compElements.append(i)
59      for i in list1:
60          if i not in compElements:
61              compElements.append(i)
```

Assign3 > 🐍 Playfair.py > ...

```python
62
63          matrix = []
64          while compElements != []:
65              matrix.append(compElements[:5])
66              compElements = compElements[5:]
67
68          return matrix
69
70    def printMatrix(matrix):
71        for row in matrix:
72            print(" ".join(row))
73        print()
74
75    def search(mat, element):
76        for i in range(5):
77            for j in range(5):
78                if(mat[i][j] == element):
79                    return i, j
80
81    def encrypt_RowRule(matr, e1r, e1c, e2r, e2c):
82        char1 = ''
83        if e1c == 4:
84            char1 = matr[e1r][0]
85        else:
86            char1 = matr[e1r][e1c+1]
87
88        char2 = ''
89        if e2c == 4:
90            char2 = matr[e2r][0]
91        else:
92            char2 = matr[e2r][e2c+1]
93
94        return char1, char2
```

```python
 96
 97    def encrypt_ColumnRule(matr, e1r, e1c, e2r, e2c):
 98        char1 = ''
 99        if e1r == 4:
100            char1 = matr[0][e1c]
101        else:
102            char1 = matr[e1r+1][e1c]
103
104        char2 = ''
105        if e2r == 4:
106            char2 = matr[0][e2c]
107        else:
108            char2 = matr[e2r+1][e2c]
109
110        return char1, char2
111
112    def encrypt_RectangleRule(matr, e1r, e1c, e2r, e2c):
113        char1 = ''
114        char1 = matr[e1r][e2c]
115
116        char2 = ''
117        char2 = matr[e2r][e1c]
118
119        return char1, char2
120
121    def encryptByPlayfairCipher(Matrix, plainList):
122        CipherText = []
123        for i in range(0, len(plainList)):
124            c1 = 0
125            c2 = 0
126            ele1_x, ele1_y = search(Matrix, plainList[i][0])
127            ele2_x, ele2_y = search(Matrix, plainList[i][1])
```

```
Assign3 > Playfair.py > main
129            if ele1_x == ele2_x:
130                c1, c2 = encrypt_RowRule(Matrix, ele1_x, ele1_y, ele2_x, ele2_y)
131                # Get 2 letter cipherText
132            elif ele1_y == ele2_y:
133                c1, c2 = encrypt_ColumnRule(Matrix, ele1_x, ele1_y, ele2_x, ele2_y)
134            else:
135                c1, c2 = encrypt_RectangleRule(
136                    Matrix, ele1_x, ele1_y, ele2_x, ele2_y)
137
138            cipher = c1 + c2
139            CipherText.append(cipher)
140        return CipherText
141
142    def main():
143        text_Plain = input("Enter the message to encrypt: ")
144        text_Plain = removeSpaces(toLowerCase(text_Plain))
145        PlainTextList = Diagraph(FillerLetter(text_Plain))
146        if len(PlainTextList[-1]) != 2:
147            PlainTextList[-1] = PlainTextList[-1]+'z'
148
149        key = input("Enter the encryption key: ")
150        key = toLowerCase(key)
151        Matrix = generateKeyTable(key, list1)
152        print("\nMatrix:")
153        printMatrix(Matrix)
154        print("Plain Text:", text_Plain)
155        CipherList = encryptByPlayfairCipher(Matrix, PlainTextList)
156        CipherText = ""
157        for i in CipherList:
158            CipherText += i
159        print("CipherText:", CipherText)
160
161    if __name__ == "__main__":
162        main()
```

OUTPUT:

```
PS C:\Users\ANUSHKA\Documents\BTech\CNS> python -u "c:\Users\ANUSHKA\Documents\BTech\CNS\Playfair.py"
Enter the message to encrypt: My name is Anushka
Enter the encryption key: anushit

Matrix:
a n u s h
i t b c d
e f g k l
m o p q r
v w x y z

Plain Text: mynameisanushka
CipherText: qvunvmcanushslhv
PS C:\Users\ANUSHKA\Documents\BTech\CNS>
```

Name: Vishal Indradev Chauhan

PRN: 2020BTECS00090

Class: Final Year - CSE

Assignment No 4

Cryptography and Network Security Lab (5CS453)

Problem Statement:Vigenère Cipher Technique

☐

The Vigenère Cipher Technique is a polyalphabetic substitution cipher that adds an extra layer of complexity to encryption by using a keyword or key phrase to determine the shifts applied to the plaintext letters.

Unlike monoalphabetic ciphers, where each letter is replaced with a fixed substitution, the Vigenère Cipher employs multiple alphabets with different shifts, making it more secure against frequency analysis.

Key Setup:

- Choose a keyword or key phrase.
- Replicate the keyword to match the length of the plaintext, repeating it as needed.
- Convert the keyword letters to their corresponding numerical values (A=0, B=1, ..., Z=25).

Encryption:

- Divide the plaintext into individual letters and convert them to numerical values.
- For each letter, determine the shift value using the corresponding keyword letter.
- Shift the plaintext letter by the calculated shift value (mod 26).
- Convert the shifted numerical value back to a letter to create the ciphertext.

Decryption:

- Divide the ciphertext into individual letters and convert them to numerical values.
- For each letter, determine the shift value using the corresponding keyword letter.
- Reverse the shift (subtract the shift value, mod 26).
- Convert the shifted numerical value back to a letter to retrieve the original plaintext.

Advantages:

- Stronger security due to polyalphabetic nature and keyword-driven shifts.
- Reduces susceptibility to frequency analysis.
- Key space increases with keyword length, enhancing security.

Disadvantages:

- Vulnerable to key length repetition (Kasiski examination) for shorter keywords.

- Security can weaken if the keyword is short or predictable.

CODE:

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


def main():
    string = input("Enter the plaintext: ").upper()
    keyword = input("Enter the keyword: ").upper()
    key = generateKey(string, keyword)
    cipher_text = cipherText(string, key)
    print("Ciphertext:", cipher_text)
    print("Original/Decrypted Text:",
          originalText(cipher_text, key))

if __name__ == "__main__":
    main()
```

OUTPUT:

```
PS C:\Users\ANUSHKA\Documents\BTech\CNS> python -u "c:\Users\ANUSHKA\Documents\BTech\CNS\vigenereCipher.py"
Enter the plaintext: anushka
Enter the keyword: jadhav
Ciphertext: JNXZHFJ
Original/Decrypted Text: ANUSHKA
PS C:\Users\ANUSHKA\Documents\BTech\CNS> []
```

<u>Name: Vishal Indradev Chauhan</u>

<u>PRN: 2020BTECS00090</u>

<u>Class: Final Year - CSE</u>

Assignment No 5

Cryptography and Network Security Lab (5CS453)

⇒**Problem Statement:Data Encryption Standard (DES)**

- **Data Encryption Standard (DES)** is one of the earliest and most widely used encryption standards for securing digital data.
- Developed in the early 1970s by IBM, it was adopted as a federal standard in the United States for the protection of sensitive data.
- It's a symmetric-key block cipher, meaning the same key is used for both encryption and decryption, and it operates on fixed-size data blocks.
- DES uses a 56-bit encryption key (originally 64 bits with 8 bits used for parity). The key length is relatively short by modern standards.
- DES uses a Feistel network structure, dividing data into halves and applying multiple rounds of operations with the encryption key.
- However, due to its vulnerability to brute-force attacks and advances in computing power, DES is now considered obsolete and has been largely replaced by more secure encryption algorithms like AES (Advanced Encryption Standard).

Code:

```python
from Crypto.Cipher import DES
from secrets import token_bytes

key = token_bytes(8)

def encrypt(msg):
    cipher = DES.new(key,DES.MODE_EAX)
    nonce = cipher.nonce
    ciphertext,tag = cipher.encrypt_and_digest(msg.encode('ascii'))
    return nonce, ciphertext,tag

def decrypt(nonce, ciphertext, tag):
    cipher = DES.new(key, DES.MODE_EAX, nonce=nonce)
    plaintext = cipher.decrypt(ciphertext)

    try:
        cipher.verify(tag)
        return plaintext.decode('ascii')
    except:
        return False
```

```python
print('\n*** Data Encryption Standard Algorithm ***')
nonce, ciphertext, tag = encrypt(input('Enter Plain Text:'))
plaintext = decrypt(nonce,ciphertext,tag)

print(f'Cipher Text is: {ciphertext}')

if not plaintext:
    print('Message is Corrupted!')
else:
    print(f'Plain Text is: {plaintext}')
```

Output:

```
*** Data Encryption Standard Algorithm ***
Enter Plain Text:Hey there! Anushka here..
Cipher Text is: b'j\xab)eZ~\x94\x01X\xdb\t\x8f\x13A\x10w\xc2\xd7wD\xb2\xe0\x10\xdc\xcb'
Plain Text is: Hey there! Anushka here..
PS C:\Users\ANUSHKA\Documents\BTech\CNS\Assign5> █
```