KADI SARVA VISHWAVIDYALAYA

LDRP INSTITUTE OF TECHNOLOGY AND RESEARCH

GANDHINAGAR

Department of

Computer Engineering and Information Technology

Subject: Cryptography and Network Security (CE603 – N)

Laboratory Manual

Prepared By

Student Name: PRAJAPATI YASH HASMUKHBHAI

Enrollment No: 19BECE30212

6CE-Division: E

INDEX

PRACTICAL - 1

AIM: Implement Caeser Cipher with Variable Key.

PROGRAM:

```
loop = "Y"
while loop == "Y" or loop == "y":
    print("1. Encrypt The Plain Text")
    print("2. Decrypt The Cipher Text")
    choice = input("Enter Your Choice : ")
    if choice == "1":
        key = input("Please Enter The Key :")
        plain_text = input("Please Enter The Plain Text : ")
        cipher_text = ""

        for i in range(len(plain_text)):
            letter = plain_text[i]
            if letter.isupper():
                cipher_text += chr((ord(letter) + int(key) - 65) % 26 + 65)
            else:
                cipher_text += chr((ord(letter) + int(key) - 97) % 26 + 97)

        print(f" Plain Text : {plain_text}")
        print(f" Cipher Text : {cipher_text}")

    elif choice =="2":
```

```
        key = input("Please Enter The Key : ")

        cipher_text = input("Please Enter The Cipher Text : ")

        plain_text = ""

        for i in range(len(cipher_text)):

            letter = cipher_text[i]


            if letter.isupper():

                plain_text += chr((ord(letter) - int(key) - 65) % 26 + 65)


            else:

                plain_text += chr((ord(letter) - int(key) - 97) % 26 + 97)


        print(f" Cipher Text : {cipher_text}")

        print(f" Plain Text : {plain_text}")


    else:

        print("Invalid Choice")


    loop=input("Do you want to continue(y/n) : ")

else :

 print(" Program is Terminating.")
```

OUTPUT:

```
1. Encrypt The Plain Text
2. Decrypt The Cipher Text
Enter Your Choice : 1
Please Enter The Key :17
Please Enter The Plain Text : ANSH
 Plain Text : ANSH
 Cipher Text : REJY
Do you want to continue(y/n) : Y
1. Encrypt The Plain Text
2. Decrypt The Cipher Text
Enter Your Choice : 2
Please Enter The Key : 17
Please Enter The Cipher Text : REJY
 Cipher Text : REJY
 Plain Text : ANSH
Do you want to continue(y/n) : n
 Program is Terminating.


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

PRACTICAL - 2

AIM: Implement the Brute Force Attack on Caeser Cipher.

PROGRAM:

```
loop = "Y"
cracked_key = 0

def display1(key, org_text):
    print(f"{key}   {org_text}")


def display2(key, org_text):
    print(f"{key}   {org_text}")


while loop == "Y" or loop == "y":
    print(" Doing ENCRYPTION : ")
    key = input(" Please Enter the Key : ")
    plain_text = input(" Please Enter the Plain Text : ")
    cipher_text = ""
    for i in range(len(plain_text)):
        letter = plain_text[i]

        if letter.isupper():
            cipher_text += chr((ord(letter) + int(key) - 65) % 26 + 65)

        else:
            cipher_text += chr((ord(letter) + int(key) - 97) % 26 + 97)

    print(f"\n Plain Text : {plain_text}")
```

```
    print(f" Cipher Text : {cipher_text}")

    print(" Attacker Doing Brute-Force Attack Using Exhaustive Key Search :")
    print(" KEY USED  PLAIN_TEXT")
    for j in range(26):
      org_text = ""
      for i in range(len(cipher_text)):
        letter = cipher_text[i]

        if letter.isupper():
          org_text += chr((ord(letter) - int(j) - 65) % 26 + 65)

        else:
          org_text += chr((ord(letter) - int(j) - 97) % 26 + 97)

      if j<=9:
        display1(j, org_text)
      else:
        display2(j, org_text)

      if plain_text == org_text:
        matched_text = org_text
        cracked_key=j

    print(f"\n Matched Text : {matched_text} \n Key Used to encrypt : {cracked_key}")
    print(" Plain Text Encrypted by Sender : {plain_text}")
    loop = input(" Do you want to continue(y/n): ")

  else:
   print(" Program is terminating.")
```

OUTPUT:

```
 Doing ENCRYPTION :
 Please Enter the Key : 17
 Please Enter the Plain Text : ANSHPARIKH

 Plain Text : ANSHPARIKH
 Cipher Text : REJYGRIZBY
 Attacker Doing Brute-Force Attack Using Exhaustive Key Search :
 KEY USED   PLAIN_TEXT
0 REJYGRIZBY
1 QDIXFQHYAX
2 PCHWEPGXZW
3 OBGVDOFWYV
4 NAFUCNEVXU
5 MZETBMDUWT
6 LYDSALCTVS
7 KXCRZKBSUR
8 JWBQYJARTQ
9 IVAPXIZQSP
10 HUZOWHYPRO
11 GTYNVGXOQN
12 FSXMUFWNPM
13 ERWLTEVMOL
14 DQVKSDULNK
15 CPUJRCTKMJ
16 BOTIQBSJLI
17 ANSHPARIKH
18 ZMRGOZQHJG
19 YLQFNYPGIF
```

```
8 JWBQYJARTQ
9 IVAPXIZQSP
10 HUZOWHYPRO
11 GTYNVGXOQN
12 FSXMUFWNPM
13 ERWLTEVMOL
14 DQVKSDULNK
15 CPUJRCTKMJ
16 BOTIQBSJLI
17 ANSHPARIKH
18 ZMRGOZQHJG
19 YLQFNYPGIF
20 XKPEMXOFHE
21 WJODLWNEGD
22 VINCKVMDFC
23 UHMBJULCEB
24 TGLAITKBDA
25 SFKZHSJACZ

 Matched Text : ANSHPARIKH
 Key Used to encrypt :  17
 Plain Text Encrypted by Sender : {plain_text}
 Do you want to continue(y/n): N
 Program is terminating.


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

PRACTICAL - 3

AIM: Implement Simple Transposition Technique.

PROGRAM:

```
import math


loop = "Y"
while loop == "Y" or loop == "y":
    print("1. Encrypt The Plain Text")
    print("2. Decrypt The Cipher Text")
    choice = input("Enter Your Choice:")
    if choice == "1":
        key = input("Please Enter The Key:")
        key = str.upper(key)
        plain_text = input("Please Enter The Plain Text:")
        cipher_text = ""
        k_indx = 0
        plain_text_len = float(len(plain_text))
        plain_text_lst = list(plain_text)
        key_lst = sorted(list(key))
        col = len(key)
        row = int(math.ceil(plain_text_len/col))
        fill_null = int((row * col) - plain_text_len)
        plain_text_lst.extend('_' * fill_null)
        matrix = [plain_text_lst[i: i + col] for i in range(0, len(plain_text_lst), col)]

        for _ in range(col):
            curr_idx = key.index(key_lst[k_indx])
            cipher_text += ''.join([row[curr_idx] for row in matrix])
            k_indx += 1
```

```
        print(f" Plain Text: {plain_text}")
        print(f" Cipher Text: {cipher_text}")

    elif choice =="2":
        key = input("Please Enter The Key:")
        cipher_text = input("Please Enter The Cipher Text:")
        plain_text = ""
        k_indx = 0
        plain_text_indx = 0
        plain_text_len = float(len(cipher_text))
        plain_text_lst = list(cipher_text)
        col = len(key)
        row = int(math.ceil(plain_text_len/col))
        key_lst = sorted(list(key))
        dec_cipher = []

        for _ in range(row):
            dec_cipher += [[None] * col]

        for _ in range(col):
            curr_idx = key.index(key_lst[k_indx])
            for j in range(row):
                dec_cipher[j][curr_idx] = plain_text_lst[plain_text_indx]
                plain_text_indx += 1
            k_indx += 1

        try:
            plain_text = ''.join(sum(dec_cipher, []))
        except TypeError:
            raise TypeError("This Program Cannot Handle Repeating Words.")
```

```
    null_count = plain_text.count('_')
    if null_count > 0:
      plain_text = plain_text[: -null_count]
    print(f"Cipher text: {cipher_text}")
    print(f"Plain text:{plain_text}")
  else:
    print("Invalid choice")
  loop=input("Do you want to continue(y/n):")

else :
  print(" Program is terminated.")
```

OUTPUT:

```
1. Encrypt The Plain Text
2. Decrypt The Cipher Text
Enter Your Choice:1
Please Enter The Key:Ansh
Please Enter The Plain Text:Parikh
 Plain Text: Parikh
 Cipher Text: Pki_ahr_
Do you want to continue(y/n):y
1. Encrypt The Plain Text
2. Decrypt The Cipher Text
Enter Your Choice:2
Please Enter The Key:Ansh
Please Enter The Cipher Text:Pki_ahr_
Cipher text: Pki_ahr_
Plain text:Parikh
Do you want to continue(y/n):n
 Program is terminated.


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

PRACTICAL - 4

AIM: Implement Simple Permutation Technique.

PROGRAM:

```
import math


loop = "Y"
while loop == "Y" or loop == "y":
    print("1. Encrypt The Plain Text")
    print("2. Decrypt The Cipher Text",end="\n\n")
    choice = input("Enter Your Choice:")
    if choice == "1":
        key = input("Please Enter The Key:")
        key = str.upper(key)
        plain_text = input("Please Enter The Plain Text:")
        cipher_text = ""
        k_indx = 0
        plain_text_len = float(len(plain_text))
        plain_text_lst = list(plain_text)
        key_lst = sorted(list(key))
        col = len(key)
        row = int(math.ceil(plain_text_len/col))
        fill_null = int((row * col) - plain_text_len)
        plain_text_lst.extend('_' * fill_null)
        matrix = [plain_text_lst[i: i + col] for i in range(0, len(plain_text_lst), col)]

        for _ in range(col):
            curr_idx = key.index(key_lst[k_indx])
            cipher_text += ''.join([row[curr_idx] for row in matrix])
            k_indx += 1
```

```
        print(f" Plain Text: {plain_text}")
        print(f" Cipher Text: {cipher_text}")
        print()
    elif choice =="2":
        key = input("Please Enter The Key:")
        cipher_text = input("Please Enter The Cipher Text:")
        plain_text = ""
        k_indx = 0
        plain_text_indx = 0
        plain_text_len = float(len(cipher_text))
        plain_text_lst = list(cipher_text)
        col = len(key)
        row = int(math.ceil(plain_text_len/col))
        key_lst = sorted(list(key))
        dec_cipher = []

        for _ in range(row):
            dec_cipher += [[None] * col]

        for _ in range(col):
            curr_idx = key.index(key_lst[k_indx])
            for j in range(row):
                dec_cipher[j][curr_idx] = plain_text_lst[plain_text_indx]
                plain_text_indx += 1
            k_indx += 1

        try:
            plain_text = ''.join(sum(dec_cipher, []))
        except TypeError:
            raise TypeError("This Program Cannot Handle Repeating Words.")
```

```
        null_count = plain_text.count('_')
        if null_count > 0:
            plain_text = plain_text[: -null_count]
        print(f"Cipher text: {cipher_text}")
        print(f"Plain text:{plain_text}")
        print()
    else:
        print("Invalid choice")
        print()
    loop=input("Do you want to continue(y/n):")
    print()

else :
    print(" Program is terminated.")
```

OUTPUT:

```
input
1. Encrypt The Plain Text
2. Decrypt The Cipher Text

Enter Your Choice:1
Please Enter The Key:ANSH
Please Enter The Plain Text:PARIKH
 Plain Text: PARIKH
 Cipher Text: PKI_AHR_

Do you want to continue(y/n):y

1. Encrypt The Plain Text
2. Decrypt The Cipher Text

Enter Your Choice:2
Please Enter The Key:ANSH
Please Enter The Cipher Text:PKI_AHR_
Cipher text: PKI_AHR_
Plain text:PARIKH

Do you want to continue(y/n):n

 Program is terminated.


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

PRACTICAL - 5

AIM: Implement the rail fence cipher with variable fence.

PROGRAM:

```
def encryptRailFence(text, key):
    rail = [['\n' for i in range(len(text))]for j 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 j in range(len(text)):
            if rail[i][j] != '\n':
                result.append(rail[i][j])
    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
        else:
            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))


print(encryptRailFence("attack at once", 2))
print(encryptRailFence("Parikh Ansh", 3))
print(encryptRailFence("defend the east wall", 3))
print(decryptRailFence("PknaihAsr h", 3))
print(decryptRailFence("atc toctaka ne", 2))
print(decryptRailFence("dnhaweedtees alf tl", 3))
```

OUTPUT:

```
input
atc toctaka ne
PknaihAsr h
dnhaweedtees alf  tl
GeeksforGeeks
attack at once
delendfthe east wal


...Program finished with exit code 0
```

PRACTICAL - 6

AIM: Implement 6 x 6 Playfair Matrix

PROGRAM:

```
loop = "Y"


def matrix(x,y,initial):
    return [[initial for i in range(x)] for j in range(y)]



def createMatrix(key):
    result=list()
    for c in key:
        if c not in result:
            result.append(c)

    for i in range(65,91):
        if chr(i) not in result:
            result.append(chr(i))

    for i in range(0,10):
        if i not in result:
            result.append(i)

    k=0
    my_matrix=matrix(6,6,0)
    for i in range(0,6):
        for j in range(0,6):
            my_matrix[i][j]=result[k]
            k+=1
```

```
    return my_matrix


def displayMatrix(matrix):
    for i in range(len(matrix)):
        for j in range(len(matrix[i])):
            print(matrix[i][j], end="\t")
        print()


def locindex(c):
    loc=list()
    for i ,j in enumerate(my_matrix):
        for k,l in enumerate(j):
            if str(c)==str(l):
                loc.append(i)
                loc.append(k)
                return loc


while loop == "Y" or loop == "y":
    print(" 1. Encrypt The Plain Text")
    print(" 2. Decrypt The Cipher Text")
    choice = input(" Enter Your Choice : ")
    if choice == "1":
        key = input("Please Enter the Key : ")
        key = key.upper()
        key=key.replace(" ","")
        plain_text = str(input(" Please Enter The Plain Text : "))
        plain_text = plain_text.upper()
        plain_text = plain_text.replace(" ","")
        my_matrix = createMatrix(key)
        print(" Playfair Matrix is as follows : \n")
        displayMatrix(my_matrix)
```

```
    i=0
    for s in range(0,len(plain_text)+1,2):
        if s<len(plain_text)-1:
            if plain_text[s]==plain_text[s+1]:
                plain_text=plain_text[:s+1]+'X'+plain_text[s+1:]

    if len(plain_text)%2!=0:
        plain_text=plain_text[:]+'X'

    print(f" Plain Text : {plain_text}")
    print(f" Cipher Text : ",end="")
    while i<len(plain_text):
        loc1=list()
        loc1=locindex(plain_text[i])
        loc2=list()
        loc2=locindex(plain_text[i+1])
        if loc1[1]==loc2[1]:
            print(f"{my_matrix[(loc1[0]+1)%6][loc1[1]]}{my_matrix[(loc2[0]+1)%6][loc2[1]]}",end='')
        elif loc1[0]==loc2[0]:
            print(f"{my_matrix[loc1[0]][(loc1[1]+1)%6]}{my_matrix[loc2[0]][(loc2[1]+1)%6]}",end='')
        else:
            print(f"{my_matrix[loc1[0]][loc2[1]]}{my_matrix[loc2[0]][loc1[1]]}",end='')
        i=i+2

elif choice =="2":
    key = input("Please Enter The Key : ")
    key = key.upper()
    key=key.replace(" ","")
    cipher_text = str(input(" Please Enter The Cipher Text : "))
    cipher_text=cipher_text.upper()
    cipher_text=cipher_text.replace(" ", "")
```

```
        my_matrix = createMatrix(key)
        print(" Playfair Matrix is as follows :")
        displayMatrix(my_matrix)
        print(f"Cipher Text : {cipher_text}")
        print(" Plain Text : ")
        i=0
        while i<len(cipher_text):
          loc1=list()
          loc1=locindex(cipher_text[i])
          loc2=list()
          loc2=locindex(cipher_text[i+1])
          if loc1[1]==loc2[1]:
            print(f"{my_matrix[(loc1[0]-1)%5][loc1[1]]}{my_matrix[(loc2[0]-1)%5][loc2[1]]}",end='')
          elif loc1[0]==loc2[0]:
            print(f"{my_matrix[loc1[0]][(loc1[1]-1)%5]}{my_matrix[loc2[0]][(loc2[1]-1)%5]}",end='')
          else:
            print(f"{my_matrix[loc1[0]][loc2[1]]}{my_matrix[loc2[0]][loc1[1]]}",end='')
          i=i+2
      else:
        print("Invalid Choice")
      print()
      loop = input("Do you want to continue(y/n) : ")
    else :
     print(" Program is Terminating.")
```

OUTPUT:

```
 1. Encrypt The Plain Text
 2. Decrypt The Cipher Text
 Enter Your Choice : 1
Please Enter the Key : ANSH
 Please Enter The Plain Text : ATTACKONLDRP
 Playfair Matrix is as follows :

A       N       S       H       B       C
D       E       F       G       I       J
K       L       M       O       P       Q
R       T       U       V       W       X
Y       Z       0       1       2       3
4       5       6       7       8       9
 Plain Text : ATTACKONLDRP
 Cipher Text : NRRNAQLHKEWK
Do you want to continue(y/n) : y
 1. Encrypt The Plain Text
 2. Decrypt The Cipher Text
 Enter Your Choice : 2
Please Enter The Key : ANSH
 Please Enter The Cipher Text : NRRNAQLHKEWK
 Playfair Matrix is as follows :
A       N       S       H       B       C
D       E       F       G       I       J
K       L       M       O       P       Q
R       T       U       V       W       X
Y       Z       0       1       2       3
4       5       6       7       8       9
Cipher Text : NRRNAQLHKEWK
 Plain Text :
ATTACKONLDRP
Do you want to continue(y/n) : n
 Program is Terminating.


...Program finished with exit code 0
```

PRACTICAL - 7

AIM: Implement n x n Hill Cipher

PROGRAM:

```
def generate(n):
   global keyMatrix
   keyMatrix= [[0] * n for i in range(n)]

   global messageVector
   messageVector= [[0] for i in range(n)]

   global cipherMatrix
   cipherMatrix= [[0] for i in range(n)]

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(messageVector,n):
        for i in range(n):
                for j in range(1):
                        cipherMatrix[i][j] = 0
                        for x in range(n):
                                cipherMatrix[i][j] += (keyMatrix[i][x] * messageVector[x][j])
                        cipherMatrix[i][j] = cipherMatrix[i][j] % 26
```

```
def HillCipher(message, key,n):
    getKeyMatrix(key,n)
    for i in range(n):
        messageVector[i][0] = ord(message[i]) % 65
    encrypt(messageVector,n)
    CipherText = []
    for i in range(n):
        CipherText.append(chr(cipherMatrix[i][0] + 65))
    print("Ciphertext: ", "".join(CipherText))

message = input("Please Enter the Message:")
key = input("Please Enter the Key:")
n = int(input("Please Enter the value of n: "))
generate(n)
HillCipher(message, key,n)
```

OUTPUT:

input

```
Please Enter the Message:ACT
Please Enter the Key:GYBNQKURP
Please Enter the value of n: 3
Ciphertext:  POH


...Program finished with exit code 0
```

PRACTICAL - 8

AIM: Implement Vigenere Cipher.

PROGRAM:

```
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))
```

```
string = input(" Please Enter the string: ")
keyword = input(" Please Enter the key: ")
key = generateKey(string, keyword)
cipher_text = cipherText(string,key)
print("Ciphertext :", cipher_text)
print("Original/Decrypted Text :",originalText(cipher_text, key))
```

OUTPUT:

```
input
 Please Enter the string: Parikh
 Please Enter the key: Ansh
Ciphertext : PZVBQG
Original/Decrypted Text : PGXOQN


...Program finished with exit code 0
```

PRACTICAL - 9

AIM: Implement the auto-key cipher.

PROGRAM:

```
dict1 = {'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}

dict2 = {0: 'A', 1: 'B', 2: 'C', 3: 'D', 4: 'E', 5: 'F', 6: 'G', 7: 'H', 8: 'I', 9: 'J',
    10: 'K', 11: 'L', 12: 'M', 13: 'N', 14: 'O',15: 'P', 16: 'Q', 17: 'R', 18: 'S', 19: 'T',
    20: 'U', 21: 'V', 22: 'W', 23: 'X', 24: 'Y', 25: 'Z'}

def generate_key(message, key):
  i = 0
  while True:
    if len(key) == len(message):
      break
    if message[i] == ' ':
      i += 1
    else:
      key += message[i]
      i += 1
  return key

def cipherText(message, key_new):
  cipher_text = ''
  i = 0
  for letter in message:
    if letter == ' ':
      cipher_text += ' '
```

```
        else:
            x = (dict1[letter]+dict1[key_new[i]]) % 26
            i += 1
            cipher_text += dict2[x]
    return cipher_text

def originalText(cipher_text, key_new):
    or_txt = ''
    i = 0
    for letter in cipher_text:
        if letter == ' ':
            or_txt += ' '
        else:
            x = (dict1[letter]-dict1[key_new[i]]+26) % 26
            i += 1
            or_txt += dict2[x]
    return or_txt

message = 'DROP BOMB ON LDRP'
key = 'FAST'
key_new = generate_key(message, key)
cipher_text = cipherText(message, key_new)
original_text = originalText(cipher_text, key_new)
print("Encrypted Text =", cipher_text)
print("Original Text =", original_text)
```

OUTPUT:

```
input
Encrypted Text = IRGI EFAQ PB XEFC
Original Text = DROP BOMB ON LDRP


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

PRACTICAL - 10

AIM: Implement Vernam Cipher.

PROGRAM:

```
loop = "Y"
while loop == "Y" or loop == "y":
  print("1. Encrypt The Plain Text")
  print("2. Decrypt The Cipher Text")
  print()
  choice = input("Enter Your Choice:")
  if choice == "1":
    key = input("Please Enter The Key: ")
    plain_text = input("Please Enter The Plain Text: ")
    cipher_text = ''
    flag = 0
    for char in plain_text:
      cipher_text += chr(ord(char)^ord(key[flag]))
      flag += 1
      if flag == len(key):
        flag = 0
    print(f" Plain text : {plain_text}")
    print(f" Cipher text : {cipher_text}")
    print()

  elif choice =="2":
    key = input("Please Enter The Key: ")
    cipher_text = input("Please Enter The Cipher Text: ")
    plain_text = ''
    flag = 0
    for char in cipher_text:
```

```
        plain_text += chr(ord(char)^ord(key[flag]))
        flag += 1
        if flag == len(key):
          flag = 0
      print(f"Cipher text : {cipher_text}")
      print(f"Plain text : {plain_text}")
    else:
      print("Invalid choice")
    loop=input("Do you want to continue(y/n): ")
    print()
else :
 print("\n Program is terminating...")
```

OUTPUT:

```
1. Encrypt The Plain Text
2. Decrypt The Cipher Text

Enter Your Choice:1
Please Enter The Key: abcd
Please Enter The Plain Text: ANSH
 Plain text : ANSH
 Cipher text :   ,0,

Do you want to continue(y/n): y

1. Encrypt The Plain Text
2. Decrypt The Cipher Text

Enter Your Choice:2
Please Enter The Key: abcd
Please Enter The Cipher Text:   ,0,
Cipher text :   ,0,
Plain text : ANSH
Do you want to continue(y/n): n


 Program is terminating...


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

PRACTICAL - 11

AIM: Implement the One Time Pad Cipher.

PROGRAM:

```
import string
import random

loop = "Y"
key=""

while loop == "Y" or loop == "y":
    print("1. Encrypt The Plain Text")
    print("2. Decrypt The Cipher Text")
    choice = input(" Enter Your Choice: ")
    if choice == "1":
        plain_text = input(" Please Enter The Plain Text: ")
        key = ''.join(random.choices(string.ascii_uppercase, k = len(plain_text)))
        cipher_text = []
        for i in range(len(plain_text)):
            x = (ord(plain_text[i]) + ord(key[i])) % 26
            x += ord('A')
            cipher_text.append(chr(x))
        cipher_text = "".join(cipher_text)
        print(f" Plain text : {plain_text}")
        print(f" Cipher text : {cipher_text}")
        print()

    elif choice =="2":
        plain_text=[]
        cipher_text = input(" Please Enter The Cipher Text: ")
```

```
            for i in range(len(cipher_text)):
                x = (ord(cipher_text[i]) - ord(key[i]) + 26) % 26
                x += ord('A')
                plain_text.append(chr(x))
            plain_text = "".join(plain_text)
            print(f" Cipher text : {cipher_text}")
            print(f" Plain text: {plain_text}")
            print()

        else:
            print("Invalid Choice")
            print()
        loop = input(" Do you want to continue(y/n):")

else :
 print(" Program is terminating.")
```

OUTPUT:

```
1. Encrypt The Plain Text
2. Decrypt The Cipher Text
 Enter Your Choice: 1
 Please Enter The Plain Text: ATTACK
 Plain text : ATTACK
 Cipher text : RDJSMG

 Do you want to continue(y/n):y
1. Encrypt The Plain Text
2. Decrypt The Cipher Text
 Enter Your Choice: 2
 Please Enter The Cipher Text: RDJSMG
 Cipher text : RDJSMG
 Plain text: ATTACK

 Do you want to continue(y/n):n
 Program is terminating.


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

PRACTICAL - 12

AIM: Implement the Cryptanalysis using Frequency analysis.

PROGRAM:

```
def printString(S, N):
    plain_text = [None] * 5
    freq = [0] * 26
    freq_sorted = [None] * 26
    used = [0] * 26
    for i in range(N):
        if S[i] != ' ':
            freq[ord(S[i]) - 65] += 1

    for i in range(26):
        freq_sorted[i] = freq[i]

    T = "ETAOINSHRDLCUMWFGYPBVKJXQZ"
    freq_sorted.sort(reverse = True)

    for i in range(5):
        ch = -1
        for j in range(26):
            if freq_sorted[i] == freq[j] and used[j] == 0:
                used[j] = 1
                ch = j
                break

        if ch == -1:
            break
```

```
        x = ord(T[i]) - 65
        x = x - ch
        curr = ""
        for k in range(N):
            if S[k] == ' ':
                curr += " "
                continue
            y = ord(S[k]) - 65
            y += x
            if y < 0:
                y += 26
            if y > 25:
                y -= 26
            curr += chr(y + 65)
        plain_text[i] = curr

    for i in range(5):
        print(plain_text[i])



S = "B TJNQMF NFTTBHF"
N = len(S)
printString(S, N)
```

OUTPUT:

```
A SIMPLE MESSAGE
B TJNQMF NFTTBHF
A SIMPLE MESSAGE
C UKORNG OGUUCIG
C UKORNG OGUUCIG


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

PRACTICAL - 13

AIM: Implement Euclidean Algorithm and Extended Euclidean Algorithm.

PROGRAM:

```
def calculate(x, y, q):
   return (x - (q*y))


def euclideanAlgorithm(a, b, s1, s2, t1, t2):
   if b == 0 :
      print(f" {a} {b} {s1} {s2} {t1} {t2} ")
      return a,s1,t1


   elif a == 0:
      print(f"0 {a} {b} 0 {s1} {s2} 1 {t1} {t2} 0")
      return b,s1,t1
   else:
      q=a//b
      r=a%b
      t=calculate(t1,t2,q)
      s=calculate(s1,s2,q)
      print(f"{q} {a} {b} {r} {s1} {s2} {s} {t1} {t2} {t}")
      a=b
      b=r
      s1=s2
      s2=s
      t1=t2
      t2=t
```

```
    x,y,z=euclideanAlgorithm(a,b,s1,s2,t1,t2)
    return x,y,z


a = int(input("Enter The First Number:"))
b = int(input("Enter The Second Number:"))
print("q r1 r2 r s1 s2 s t1 t2 t")
print("------------------------------------------------------------")
gcdValue, s, t = euclideanAlgorithm(a, b, 1, 0, 0, 1)
print(f" Using Extended Euclidean Algorithm : \n GCD({a},{b}) = {gcdValue}")
print(f" s = {s}, t = {t} [a*s + b*t = GCD(a,b)]")
```

OUTPUT:

```
Enter The First Number:70
Enter The Second Number:50
q r1 r2 r s1 s2 s t1 t2 t
------------------------------------------------------------
1 70 50 20 1 0 1 0 1 -1
2 50 20 10 0 1 -2 1 -1 3
2 20 10 0 1 -2 5 -1 3 -7
 10 0 -2 5 3 -7
 Using Extended Euclidean Algorithm :
 GCD(70,50) = 10
 s = -2, t = 3 [a*s + b*t = GCD(a,b)]


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

PRACTICAL - 14

AIM: Implement Diffie-Hellman Algorithm for Key Exchange with Small Number

PROGRAM:

```
q = int(input("Enter the value of 'q'(Prime Number): "))
alpha = int(input("Enter the value of α(primitive root of 'q' and α<q'): "))

print(f"\n q={q}")
print(f" α(Primitive Root of \'q\'')={alpha}")
Xa = int(input("Enter the private key of Sender(Xa): "))
print(f"\n The Private Key Xa for Sender : {Xa}")
Ya = int(pow(alpha,Xa,q))
print(f" The Public Key Ya for Sender : {Ya} ")
Xb = int(input("Enter The Private Key Of The Receiver(Xb): "))
print(f"\n The Private Key Xb For Receiver : {Xb}")
Yb = int(pow(alpha,Xb,q))
print(f" The Private Key Yb For Sender : {Yb}")
Ka = int(pow(Yb,Xa,q))
Kb =int(pow(Ya,Xb,q))
print(f"\n The Public Key Ka For Sender : {Ka} ")
print(f" The Public Key Kb For Sender : {Kb} ")
print(" Secret Key of Both parties are Same.")
print(" Key Exchange is Successful.")
```

OUTPUT:

```
Enter the value of 'q'(Prime Number): 7
Enter the value of α(primitive root of 'q' and α<q'): 5

 q=7
 α(Primitive Root of 'q'')=5
Enter the private key of Sender(Xa): 3

 The Private Key Xa for Sender : 3
 The Public Key Ya for Sender : 6
Enter The Private Key Of The Receiver(Xb): 6

 The Private Key Xb For Receiver : 6
 The Private Key Yb For Sender : 1

 The Public Key Ka For Sender : 1
 The Public Key Kb For Sender : 1
 Secret Key of Both parties are Same.
 Key Exchange is Successful.


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

PRACTICAL - 15

AIM: Implement RSA Algorithm with Small Number

PROGRAM:

```
def GCD(a, b):
    if b == 0 :
        return a

    elif a == 0:
        return b

    else:
        r=a%b
        a=b
        b=r
        x=GCD(a,b)
        return x

def modInverse(a, m):
    for x in range(1, m):
        if (((a%m) * (x%m)) % m == 1):
            return x
    return -1

p = int(input(" Enter The 'p'(Prime): "))
q = int(input(" Enter The 'q'(Prime): "))

n = p*q
phi_n =( p-1)*(q-1)
e = int(input(f" Enter The 'e'(GCD(e,{phi_n})=1 and 1<e<{phi_n})): "))
```

```
while e < phi_n:
    if (GCD(e, phi_n)==1):
        break
    else:
        e+=1

k = 2
d = modInverse(e,phi_n)

print(f" 'd' is {d}")

privateKey = set([d,n])
publicKey = set([e,n])

print(f"\n The Private Key is {privateKey}")
print(f" The Public Key is {publicKey}")

M = int(input(f" Enter The Message To Be Encrypted(M<{n}): "))

print("\n\n\n\n Encryption:")

C = int(pow(M,e,n))

print(f" Plain Text = {M}")
print(f" Cipher Text = {C}")

print("\n DECRYPTION:")

M1 = pow(C,d,n)
```

```
print(f" Cipher Text = {C}")

print(f" Plain Text = {M1}")
```

OUTPUT:

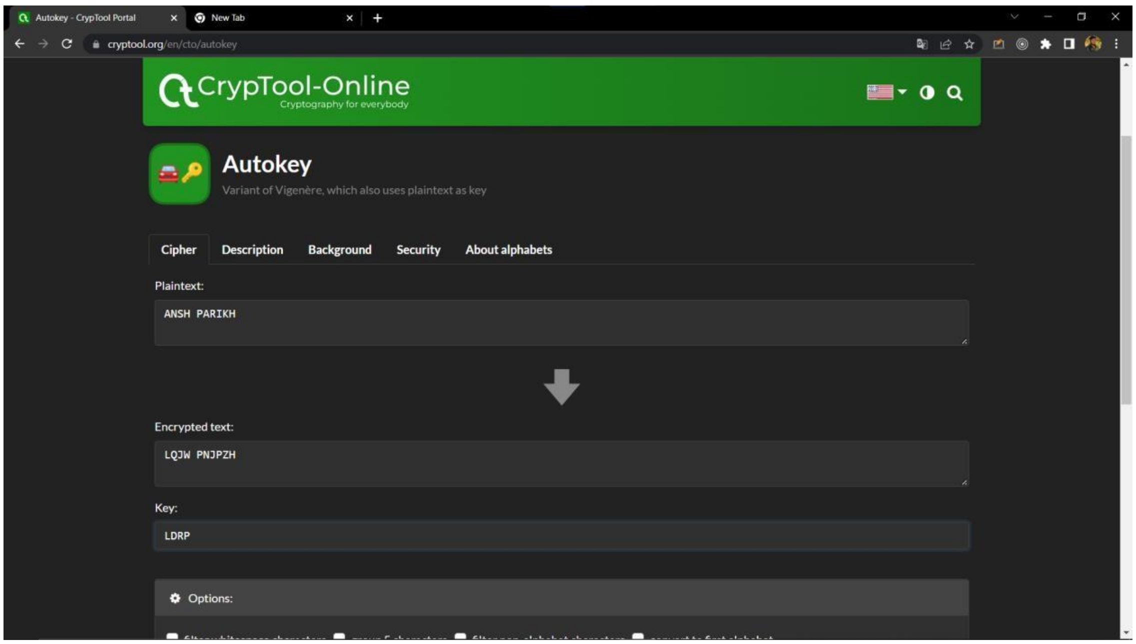
```
Enter The 'p'(Prime): 3
Enter The 'q'(Prime): 11
Enter The 'e'(GCD(e,20)=1 and 1<e<20)): 7
'd' is 3

The Private Key is {33, 3}
The Public Key is {33, 7}
Enter The Message To Be Encrypted(M<33): 17




Encryption:
Plain Text = 17
Cipher Text = 8

DECRYPTION:
Cipher Text = 8
Plain Text = 17


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

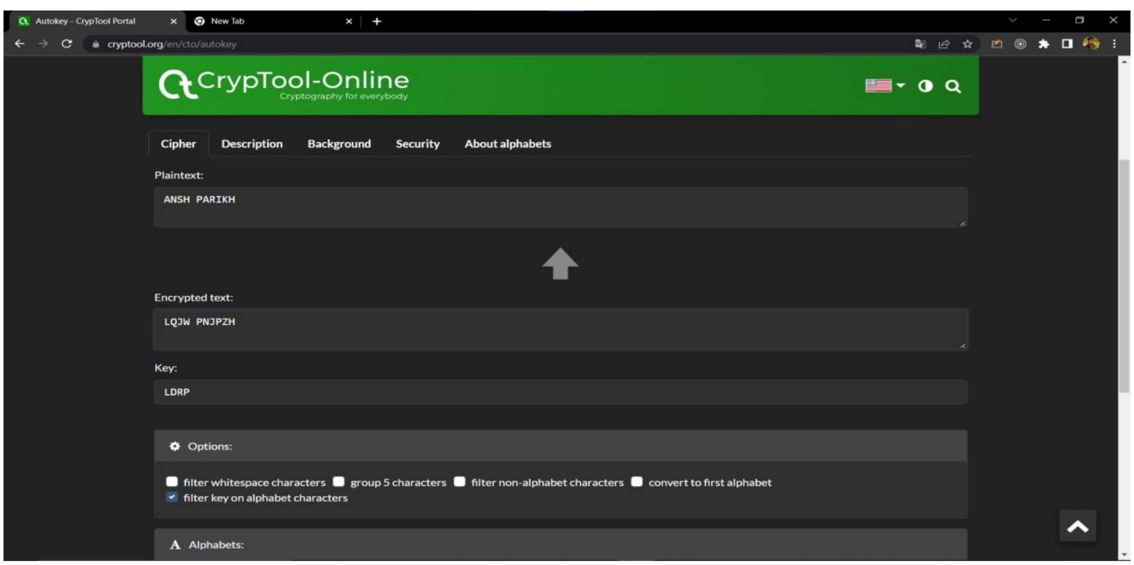
PRACTICAL - 16

AIM: STUDY VARIOUS ENCRYPTION/DECRYPTION TOOLS AVAILABLE ONLINE (EG. 'www.cryptool.org').

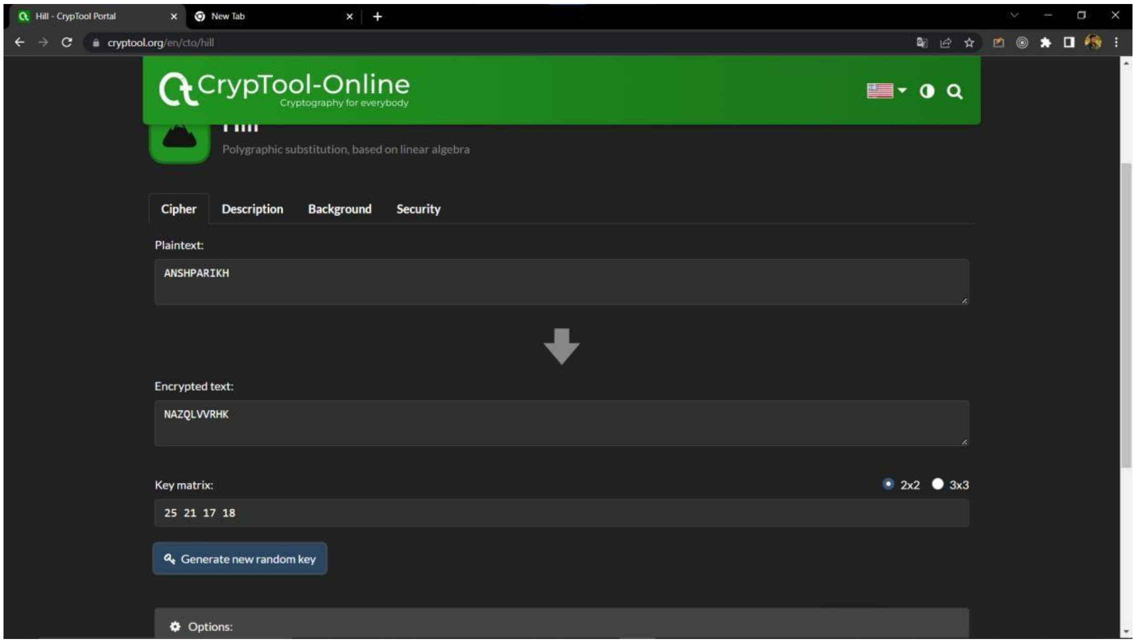
PROGRAM WITH OUTPUT:

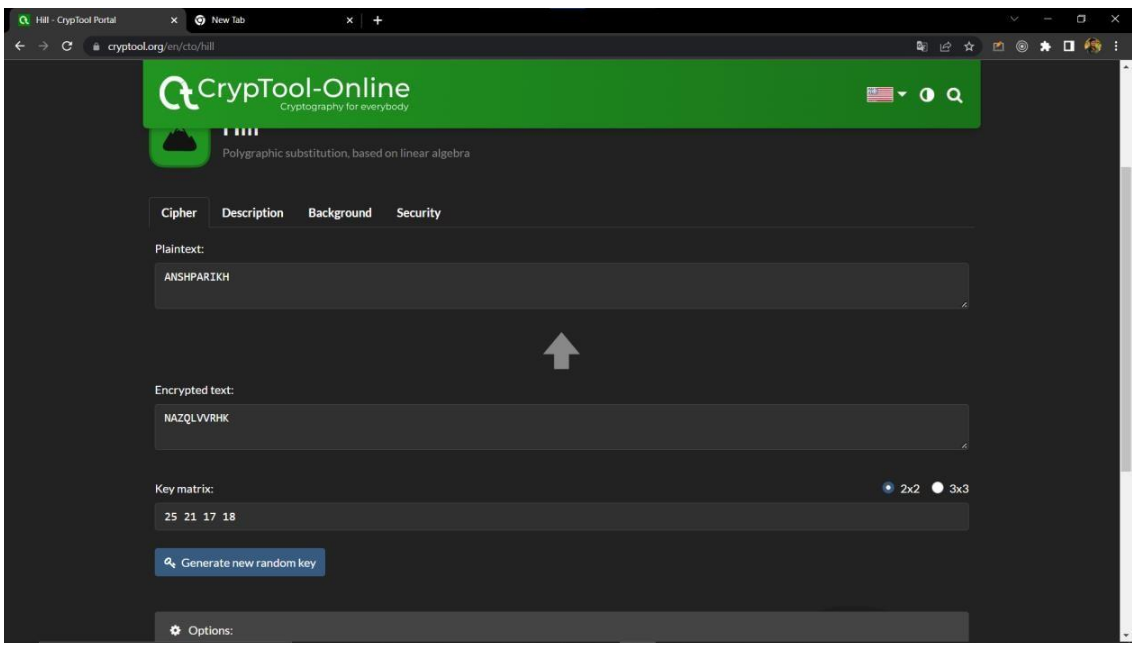
1. Autokey (Variant Of Vigenère, Which Also Uses Plain Text)



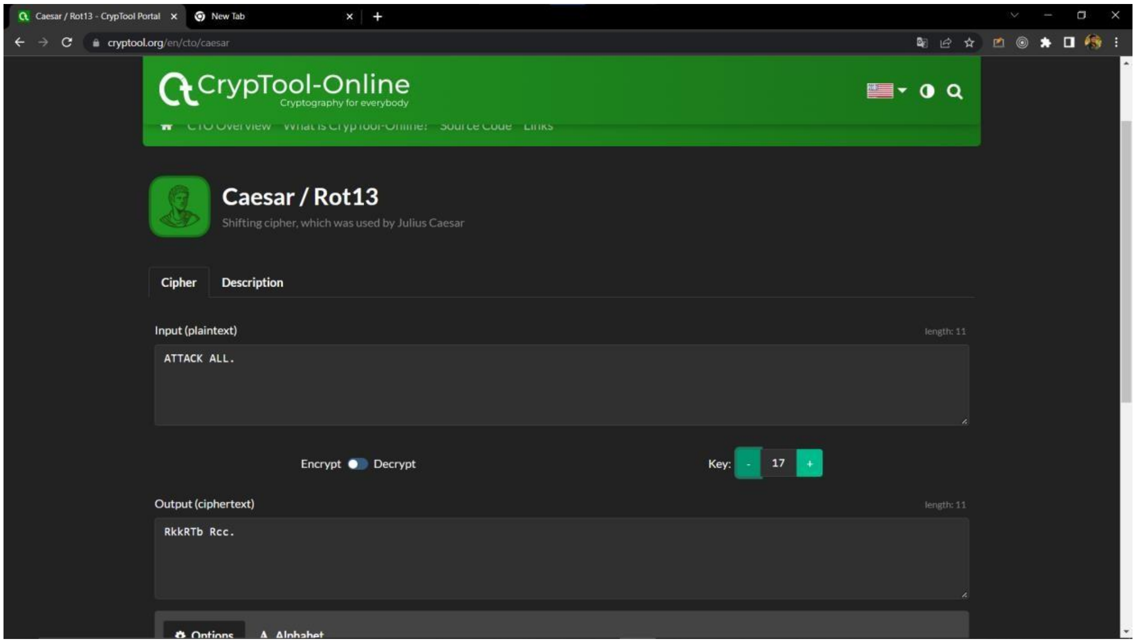


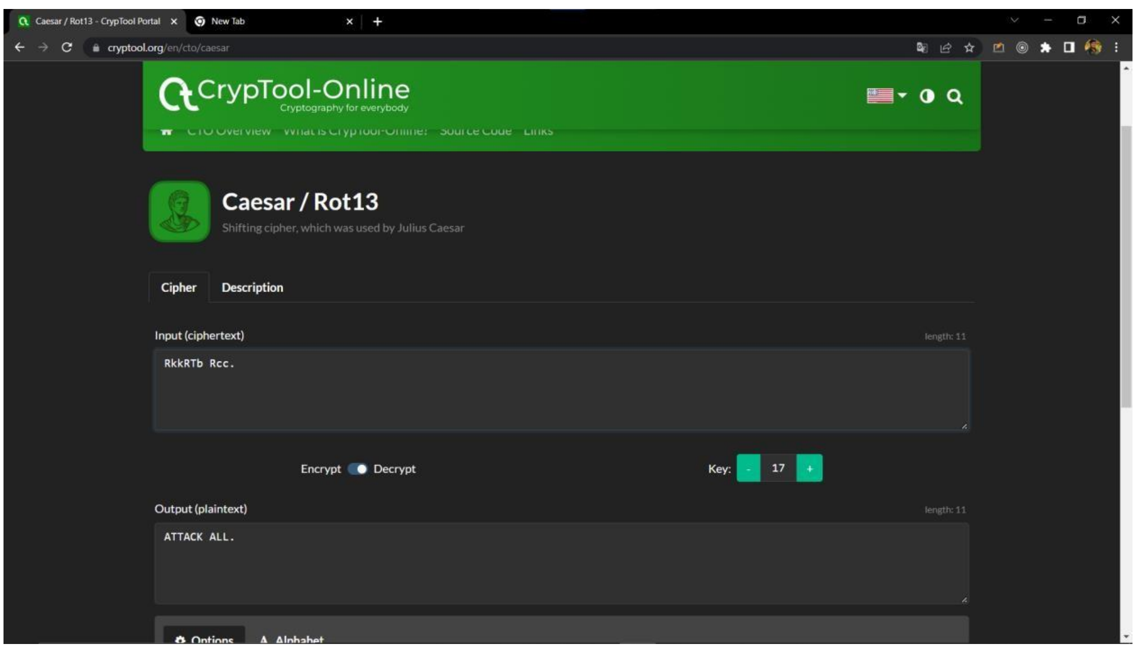
2. Hill (Polygraphic Substitution, Based On Linear Algebra)



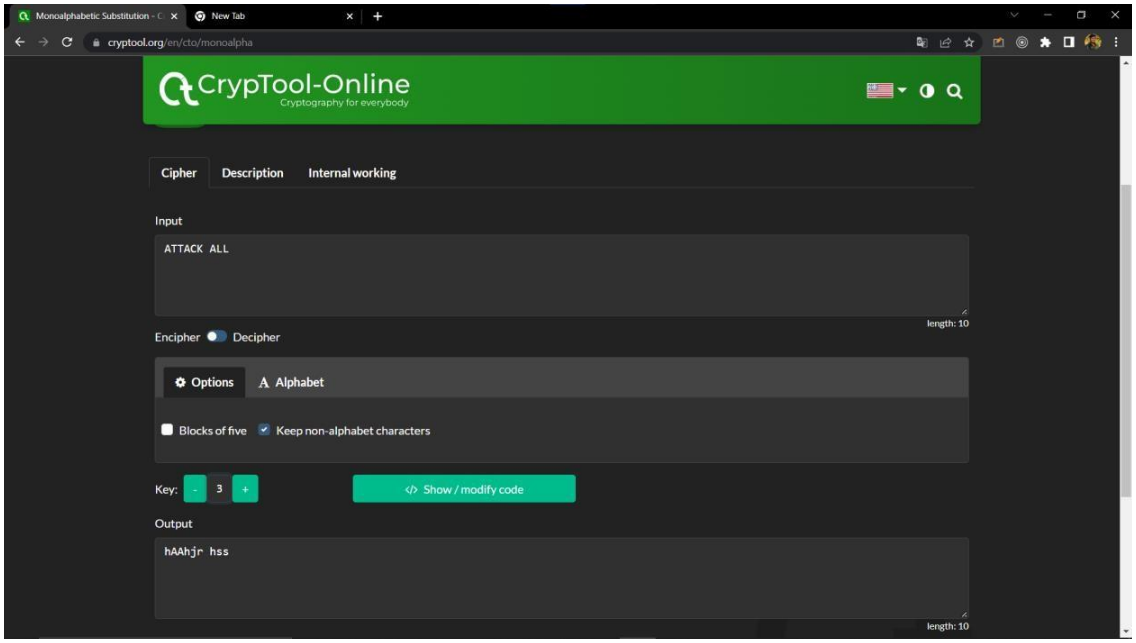


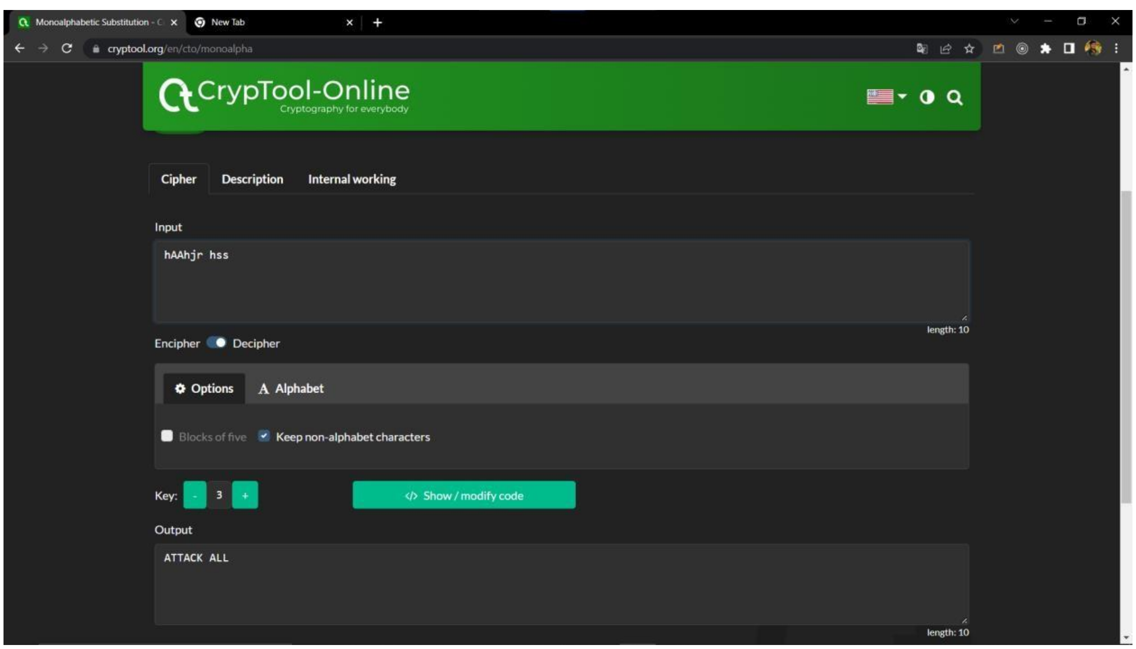
3. Caesar/Rot13 (Shifting Cipher, Which Was Used By Julius Caesar)



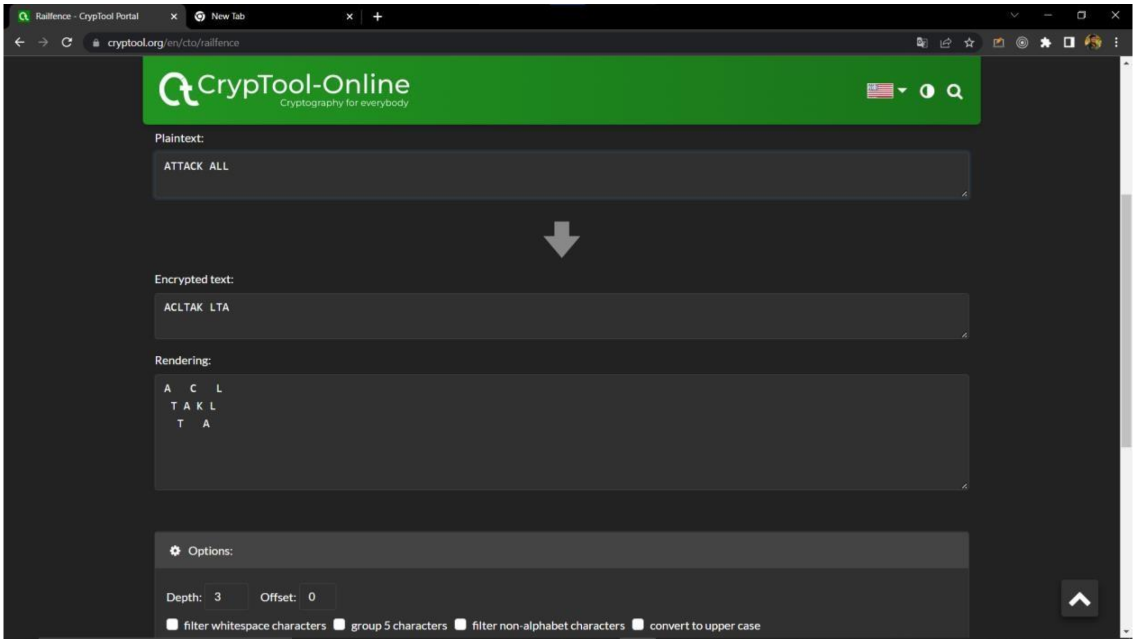


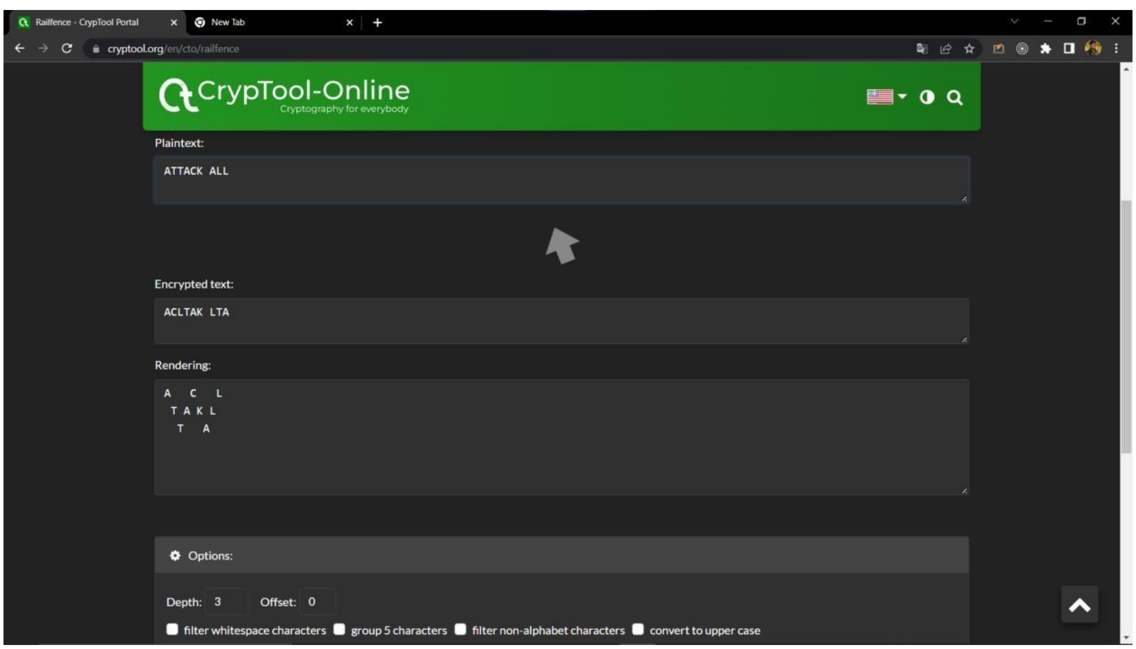
4. Monoalphabetic Substitution (Cipher That Replaces Letters With Letters/Characters)





5. Rail Fence (Transposition Cipher That Uses A Railfence Pattern)





6. XOR (Single Bits Are XORed (Typical Component Of More Complex Ciphers))