1. Implementation of Caesar Cipher

Encryption:

```
def encrypt_text(plaintext,n):
    ans = ""
    # iterate over the given text
    for i in range(len(plaintext)):
        ch = plaintext[i]

        # check if space is there then simply add space
        if ch==" ":
            ans+=" "
        # check if a character is uppercase then encrypt it accordingly
        elif (ch.isupper()):
            ans += chr((ord(ch) + n-65) % 26 + 65)
        # check if a character is lowercase then encrypt it accordingly

        else:
            ans += chr((ord(ch) + n-97) % 26 + 97)

    return ans

plaintext = "HELLO EVERYONE"
n = 1
print("Plain Text is : " + plaintext)
print("Shift pattern is : " + str(n))
print("Cipher Text is : " + encrypt_text(plaintext,n))
```

Output :

Plain Text is: HELLO EVERYONE

Shift pattern is: 1

Cipher Text is: IFMMP FWFSZPOF

2. **Brute Force Approach to Break Caesar Cipher**

```
message = 'UDZ WKLV VLGH' #encrypted message
Letters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

```
for key in range(len(Letters)):
  translated = ''
  for ch in message:
    if ch in Letters:
      num = Letters.find(ch)
      num = num - key
      if num < 0:
        num = num + len(Letters)
      translated = translated + Letters[num]
    else:
      translated = translated + ch
  print('Hacking key is %s: %s' % (key, translated))
```

Output:

```
Hacking key 0: UDZ WKLV VLGH
Hacking key 1: TCY VJKU UKFG
Hacking key 2: SBX UIJT TJEF
Hacking key 3: RAW THIS SIDE
Hacking key 4: QZV SGHR RHCD
Hacking key 5: PYU RFGQ QGBC
Hacking key 6: OXT QEFP PFAB
Hacking key 7: NWS PDEO OEZA
Hacking key 8: MVR OCDN NDYZ
Hacking key 9: LUQ NBCM MCXY
Hacking key 10: KTP MABL LBWX
Hacking key 11: JSO LZAK KAVW
Hacking key 12: IRN KYZJ JZUV
Hacking key 13: HQM JXYI IYTU
Hacking key 14: GPL IWXH HXST
Hacking key 15: FOK HVWG GWRS
Hacking key 16: ENJ GUVF FVQR
Hacking key 17: DMI FTUE EUPQ
Hacking key 18: CLH ESTD DTOP
Hacking key 19: BKG DRSC CSNO
```

Hacking key 20: AJF CQRB BRMN
Hacking key 21: ZIE BPQA AQLM
Hacking key 22: YHD AOPZ ZPKL
Hacking key 23: XGC ZNOY YOJK
Hacking key 24: WFB YMNX XNIJ
Hacking key 25: VEA XLMW WMHI

3. Implementation of Caesar Cipher – Decryption

```
def decrypt():

    #enter your encrypted message(string) below
    encrypted_message = input("Enter the message i.e to be decrypted:
").strip()

    letters="abcdefghijklmnopqrstuvwxyz"

    #enter the key value to decrypt
    k = int(input("Enter the key to decrypt: "))
    decrypted_message = ""

    for ch in encrypted_message:

        if ch in letters:
            position = letters.find(ch)
            new_pos = (position - k) % 26
            new_char = letters[new_pos]
            decrypted_message += new_char
        else:
            decrypted_message += ch
    print("Your decrypted message is:\n")
    print(decrypted_message)

decrypt()
```

Output:

Enter the message i.e to be decrypted: KYZJ ZJ R MVIP TFFC DVJJRXV

Enter the key to decrypt: 17

Your decrypted message is:

THIS IS A VERY COOL MESSAGE

4. Play Fair Cipher

```
# Create a 5x5 matrix using a secret key
def create_matrix(key):
    key = key.upper()
    matrix = [[0 for i in range (5)] for j in range(5)]
    letters_added = []
    row = 0
    col = 0
    # add the key to the matrix
    for letter in key:
        if letter not in letters_added:
            matrix[row][col] = letter
            letters_added.append(letter)
        else:
            continue
        if (col==4):
            col = 0
            row += 1
        else:
            col += 1
    #Add the rest of the alphabet to the matrix
    # A=65 ... Z=90
    for letter in range(65,91):
        if letter==74: # I/J are in the same position
                continue
        if chr(letter) not in letters_added: # Do not add repeated letters
            letters_added.append(chr(letter))

    #print (len(letters_added), letters_added)
    index = 0
```

```
    for i in range(5):
        for j in range(5):
            matrix[i][j] = letters_added[index]
            index+=1
    return matrix
```

Code to separate same letters.

```
#Add fillers if the same letter is in a pair

def separate_same_letters(message):

        index = 0

        while (index<len(message)):

                l1 = message[index]

                if index == len(message)-1:

                        message = message + 'X'

                        index += 2

                        continue

                l2 = message[index+1]

                if l1==l2:

                        message = message[:index+1] + "X" + message[index+1:]

                        index +=2

return message
```

Code to encrypt and decrypt a message

```
#Return the index of a letter in the matrix

#This will be used to know what rule (1-4) to apply
```

```
def indexOf(letter,matrix):
    for i in range (5):
        try:
            index = matrix[i].index(letter)
            return (i,index)
        except:
            continue
#Implementation of the playfair cipher
#If encrypt=True the method will encrypt the message
# otherwise the method will decrypt
def playfair(key, message, encrypt=True):
    inc = 1
    if encrypt==False:
        inc = -1
    matrix = create_matrix(key)
    message = message.upper()
    message = message.replace(' ','')
    message = separate_same_letters(message)
    cipher_text=' '
    for (l1, l2) in zip(message[0::2], message[1::2]):
        row1,col1 = indexOf(l1,matrix)
        row2,col2 = indexOf(l2,matrix)
        if row1==row2: #Rule 2, the letters are in the same row
```

```
                cipher_text += matrix[row1][(col1+inc)%5] +
            matrix[row2][(col2+inc)%5]

            elif col1==col2:# Rule 3, the letters are in the same column

                cipher_text += matrix[(row1+inc)%5][col1] +
            matrix[(row2+inc)%5][col2]

            else: #Rule 4, the letters are in a different row and column

                cipher_text += matrix[row1][col2] + matrix[row2][col1]

                return cipher_text

if __name__=='__main__':

    # a sample of encryption and decryption

    print ('Encripting')

    print ( playfair('secret', 'my secret message'))

    print ('Decrypting')

    print ( playfair('secret', 'LZECRTCSITCVAHBT', False))
```

HiLL Cipher

```
    import numpy
    def create_matrix_from(key):
        m=[[0] * 3 for i in range(3)]
        for i in range(3):
            for j in range(3):
                m[i][j] = ord(key[3*i+j]) % 65

        return m

    # C = P*K mod 26
    def encrypt(P, K):
        C=[0,0,0]
        C[0] = (K[0][0]*P[0] + K[1][0]*P[1] + K[2][0]*P[2]) % 26
```

```
        C[1] = (K[0][1]*P[0] + K[1][1]*P[1] + K[2][1]*P[2]) % 26
        C[2] = (K[0][2]*P[0] + K[1][2]*P[1] + K[2][2]*P[2]) % 26
        return C

def Hill(message, K):
        cipher_text = []
        #Transform the message 3 characters at a time
        for i in range(0,len(message), 3):
                P=[0, 0, 0]
                #Assign the corresponding integer value to each letter
                for j in range(3):
                        P[j] = ord(message[i+j]) % 65
                #Encript three letters
                C = encrypt(P,K)
                #Add the encripted 3 letters to the final cipher text
                for j in range(3):
                        cipher_text.append(chr(C[j] + 65))
                #Repeat until all sets of three letters are processed.

        #return a string

        return "".join(cipher_text)


def MatrixInverse(K):

        det = int(numpy.linalg.det(K))

        det_multiplicative_inverse = pow(det, -1, 26)

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

        for i in range(3):

                for j in range(3):

                        Dji = K

                        Dji = numpy.delete(Dji, (j), axis=0)
```

```
                    Dji = numpy.delete(Dji, (i), axis=1)
                    det = Dji[0][0]*Dji[1][1] - Dji[0][1]*Dji[1][0]
                    K_inv[i][j] = (det_multiplicative_inverse * pow(-1,i+j) *
                det) % 26
        return K_inv


if __name__ == "__main__":
        message = "MYSECRETMESSAGE"
        key = "RRFVSVCCT"
        #Create the matrix K that will be used as the key
        K = create_matrix_from(key)
        print(K)
        # C = P * K mod 26
        cipher_text = Hill(message, K)
        print ('Cipher text: ', cipher_text)
        # Decrypt
        # P = C * K^-1 mod 26
        K_inv = MatrixInverse(K)
        plain_text = Hill(cipher_text, K_inv)
        print ('Plain text: ', plain_text)
# K x K^-1 equals the identity matrix
M = (numpy.dot(K,K_inv))
for i in range(3):
```

```
            for j in range(3):
                M[i][j] = M[i][j] % 26
    print(M)
```

Vigenere Cipher

```
# This function generates the key in a cyclic manner until
# it's length isn't equal to the length of original text
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))

# This function returns the encrypted text generated with help of the 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))

# This function decrypts the encrypted text and returns the original 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))
```

```
# Driver code
if __name__ == "__main__":
    string = "GEEKSFORGEEKS"
    keyword = "AYUSH"
    key = generateKey(string, keyword)
    cipher_text = cipherText(string,key)
    print("Ciphertext :", cipher_text)
    print("Original/Decrypted Text :",originalText(cipher_text, key))
```

Rail Fence

```
# function to encrypt a message
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

    # place the marker
    rail[row][col] = '*'
    col += 1

    # find the next row
    # using direction flag
    if dir_down:
        row += 1
    else:
        row -= 1

# now we can construct the
# fill the rail matrix
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

# now read the matrix in
# zig-zag manner to construct
# the resultant text
result = []
row, col = 0, 0
for i in range(len(cipher)):

    # check the direction of flow
    if row == 0:
        dir_down = True
    if row == key-1:
        dir_down = False
```

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

        # find the next row using
        # direction flag
        if dir_down:
            row += 1
        else:
            row -= 1
    return("".join(result))

# Driver code
if __name__ == "__main__":
    print(encryptRailFence("attack at once", 2))
    print(encryptRailFence("GeeksforGeeks ", 3))
    print(encryptRailFence("defend the east wall", 3))

    # Now decryption of the
    # same cipher-text
    print(decryptRailFence("GsGsekfrek eoe", 3))
    print(decryptRailFence("atc toctaka ne", 2))
    print(decryptRailFence("dnhaweedtees alf  tl", 3))
```

Row Columnlar Transposition

```
import math

key = "HACK"

# Encryption
def encryptMessage(msg):
    cipher = ""

    # track key indices
    k_indx = 0
```

```
    msg_len = float(len(msg))
    msg_lst = list(msg)
    key_lst = sorted(list(key))

    # calculate column of the matrix
    col = len(key)

    # calculate maximum row of the matrix
    row = int(math.ceil(msg_len / col))

    # add the padding character '_' in empty
    # the empty cell of the matix
    fill_null = int((row * col) - msg_len)
    msg_lst.extend('_' * fill_null)

    # create Matrix and insert message and
    # padding characters row-wise
    matrix = [msg_lst[i: i + col]
              for i in range(0, len(msg_lst), col)]

    # read matrix column-wise using key
    for _ in range(col):
        curr_idx = key.index(key_lst[k_indx])
        cipher += ''.join([row[curr_idx]
                           for row in matrix])
        k_indx += 1

    return cipher

# Decryption
def decryptMessage(cipher):
    msg = ""

    # track key indices
    k_indx = 0

    # track msg indices
    msg_indx = 0
    msg_len = float(len(cipher))
    msg_lst = list(cipher)
```

```
# calculate column of the matrix
col = len(key)

# calculate maximum row of the matrix
row = int(math.ceil(msg_len / col))

# convert key into list and sort
# alphabetically so we can access
# each character by its alphabetical position.
key_lst = sorted(list(key))

# create an empty matrix to
# store deciphered message
dec_cipher = []
for _ in range(row):
    dec_cipher += [[None] * col]

# Arrange the matrix column wise according
# to permutation order by adding into new matrix
for _ in range(col):
    curr_idx = key.index(key_lst[k_indx])

    for j in range(row):
        dec_cipher[j][curr_idx] = msg_lst[msg_indx]
        msg_indx += 1
    k_indx += 1

# convert decrypted msg matrix into a string
try:
    msg = ''.join(sum(dec_cipher, []))
except TypeError:
    raise TypeError("This program cannot",
                    "handle repeating words.")

null_count = msg.count('_')

if null_count > 0:
    return msg[: -null_count]
```

```
    return msg

# Driver Code
msg = "Geeks for Geeks"

cipher = encryptMessage(msg)
print("Encrypted Message: {}".
              format(cipher))

print("Decryped Message: {}".
       format(decryptMessage(cipher)))
```

DES

```
pip install pycrypto
pip install base32hex
#Encryption

import base32hex
import hashlib
from Crypto.Cipher import DES
password = "Password"
salt = '\x28\xAB\xBC\xCD\xDE\xEF\x00\x33'
key = password + salt
m = hashlib.md5(key)
key = m.digest()
(dk, iv) =(key[:8], key[8:])
crypter = DES.new(dk, DES.MODE_CBC, iv)
plain_text= "I see you"
print("The plain text is : ",plain_text)
plain_text += '\x00' * (8 - len(plain_text) % 8)
ciphertext = crypter.encrypt(plain_text)
encode_string= base32hex.b32encode(ciphertext)
print("The encoded string is : ",encode_string)
#Decryption
```

```
import base32hex
import hashlib
from Crypto.Cipher import DES
password = "Password"
salt = '\x28\xAB\xBC\xCD\xDE\xEF\x00\x33'
key = password + salt
m = hashlib.md5(key)
key = m.digest()
(dk, iv) =(key[:8], key[8:])
crypter = DES.new(dk, DES.MODE_CBC, iv)
encrypted_string='UH562EGM8RCHHTOUC5CTRS59OG======'
print("The ecrypted string is : ",encrypted_string)
encrypted_string=base32hex.b32decode(encrypted_string)
decrypted_string = crypter.decrypt(encrypted_string)
print("The decrypted string is : ",decrypted_string)
```

RSA

```
pip install rsa
import rsa
```

```
def generateKeys():
    (publicKey, privateKey) = rsa.newkeys(1024)
    with open('keys/publcKey.pem', 'wb') as p:
        p.write(publicKey.save_pkcs1('PEM'))
     with open('keys/privateKey.pem', 'wb') as p:
        p.write(privateKey.save_pkcs1('PEM'))
```

```
def loadKeys():
    with open('keys/publicKey.pem', 'rb') as p:
        publicKey = rsa.PublicKey.load_pkcs1(p.read())
    with open('keys/privateKey.pem', 'rb') as p:
        privateKey = rsa.PrivateKey.load_pkcs1(p.read())
    return privateKey, publicKey
```

```
def encrypt(message, key):
```

```
    return rsa.encrypt(message.encode('ascii'), key)
```

```
def decrypt(ciphertext, key):
    try:
        return rsa.decrypt(ciphertext,
key).decode('ascii')
    except:
        return False
```

```
def sign(message, key):
    return rsa.sign(message.encode('ascii'), key, 'SHA-1')
```

```
def verify(message, signature, key):
    try:
        return rsa.verify(message.encode('ascii'), signature,
key,) == 'SHA-1'
    except:
        return False
```

```
generateKeys()
publicKey, privateKey =load_keys()
```

```
message = input('Write your message here:')
ciphertext = encrypt(message, publicKey)
```

```
signature = sign(message, privateKey)
```

```
text = decrypt(ciphertext, privateKey)
```

```
print(f'Cipher text: {ciphertext}')
print(f'Signature: {signature}')
```

```
if text:
    print(f'Message text: {text}')
else:
```

```
    print(f'Unable to decrypt the message.')
```

```
if text:
    print(f'Message text: {text}')
else:
    print(f'Unable to decrypt the message.')
```

Diffie Hellman

```
from random import randint

if __name__ == '__main__':

    # Both the persons will be agreed upon the
    # public keys G and P
    # A prime number P is taken
    P = 23

    # A primitive root for P, G is taken
    G = 9


    print('The Value of P is :%d'%(P))
    print('The Value of G is :%d'%(G))

    # Alice will choose the private key a
    a = 4
    print('The Private Key a for Alice is :%d'%(a))

    # gets the generated key
    x = int(pow(G,a,P))

    # Bob will choose the private key b
    b = 3
    print('The Private Key b for Bob is :%d'%(b))

    # gets the generated key
    y = int(pow(G,b,P))


    # Secret key for Alice
    ka = int(pow(y,a,P))

    # Secret key for Bob
    kb = int(pow(x,b,P))
```

```
    print('Secret key for the Alice is : %d'%(ka))
    print('Secret Key for the Bob is : %d'%(kb))
```

Output:

```
The value of P : 23

The value of G : 9

The private key a for Alice : 4

The private key b for Bob : 3

Secret key for the Alice is : 9

Secret Key for the Bob is : 9
```