Aim,theory, program with comments, simulation output,result

COVER PAGE WITH USNS GOES HERE

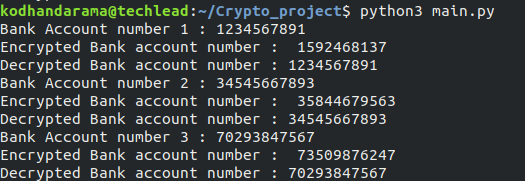
Python code for Rail-fence-cipher

|  |
| --- |
| # Rail Fence Cipher Encryption function     def RailFenceEncrypt(plain\_text, key):    """create the matrix to cipher """  # no of columns = length of plain text   # no of rows = selected key   """filling the rail matrix initialising with '\n'"""     rail = [['\n' for i in range(len(plain\_text))]   for j in range(key)]    # to find the vertical direction in which the text has to be encrypted  vertical\_direction = False  row, col = 0, 0    for i in range(len(plain\_text)):    #check the vertical direction and reverse it if  #we have reached the top or bottom rail     if (row == 0) or (row == key - 1):  vertical\_direction = not vertical\_direction    # Enter one element of plain\_text into the cipher matrix    rail[row][col] = plain\_text[i]  col += 1    """find the next row using direction flag"""    if vertical\_direction:  row += 1  else:  row -= 1  # now we can construct the cipher   # using the rail matrix  encrypted\_text = []  for i in range(key):  for j in range(len(plain\_text)):  if rail[i][j] != '\n':  encrypted\_text.append(rail[i][j])  return("" . join(encrypted\_text)) |
|  |
|  |

|  |
| --- |
| # Rail Fence Cipher Decryption function  def RailFenceDecrypt(cipher, key):     """create the matrix to cipher"""   # no of rows = plain text key   # no of columns = length(cipher)    # filling the rail matrix to distinguish filled spaces   #from blank ones    rail = [['\n' for i in range(len(cipher))]   for j in range(key)]     # To find the vertical direction in   #which to decrypt the cipher  vertical\_direction = None  row, col = 0, 0    # mark the places with '\*'   for i in range(len(cipher)):   if row == 0:   vertical\_direction = True  if row == key - 1:   vertical\_direction = False    # place the marker because this is a marked space   rail[row][col] = '\*'  col += 1    # find the next row using direction flag   if vertical\_direction:   row += 1  else:   row -= 1   """ Now we can construct the rail matrix based on the marked spaces"""  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, the marked spaces are read to give the decrypted text """  decrypted\_text = []   row, col = 0, 0  for i in range(len(cipher)):     # check the direction of flow   if row == 0:   vertical\_direction = True  if row == key-1:   vertical\_direction = False    # place the marker   if (rail[row][col] != '\*'):   decrypted\_text.append(rail[row][col])   col += 1    # find the next row using   # direction flag   if vertical\_direction:   row += 1  else:   row -= 1  return("".join(decrypted\_text)) |

|  |
| --- |
| # main.py  “”” This is the driver code that tests the encryption and decryption ”””  from rail\_encrypt import RailFenceEncrypt from rail\_decrypt import RailFenceDecrypt  print("Bank Account number 1 :","1234567891")  Bank\_account\_no1\_encrypt=RailFenceEncrypt("1234567891",3) print("Encrypted Bank account number : ",Bank\_account\_no1\_encrypt)  Bank\_account\_no1\_decrypt=RailFenceDecrypt(Bank\_account\_no1\_encrypt,3) print("Decrypted Bank account number :",Bank\_account\_no1\_decrypt)   print("Bank Account number 2 :","34545667893")  Bank\_account\_no2\_encrypt=RailFenceEncrypt("34545667893",3) print("Encrypted Bank account number : ",Bank\_account\_no2\_encrypt)  Bank\_account\_no2\_decrypt=RailFenceDecrypt(Bank\_account\_no2\_encrypt,3) print("Decrypted Bank account number :",Bank\_account\_no2\_decrypt)   print("Bank Account number 3 :","70293847567")  Bank\_account\_no3\_encrypt=RailFenceEncrypt("70293847567",3) print("Encrypted Bank account number : ",Bank\_account\_no3\_encrypt)  Bank\_account\_no3\_decrypt=RailFenceDecrypt(Bank\_account\_no3\_encrypt,3) print("Decrypted Bank account number :",Bank\_account\_no3\_decrypt) |

**Simulation Results**

****