**IS Group A Practical 1**

All theory with code available here:

<https://onlinesmarttrainer.blogspot.com/p/experiment2-xor-string-with-127-aim.html>

**IS Group A Practical 2**

**(IMPLEMENTATION OF CAESAR CIPHER)**

**Title:** Transposition technique – Caesar Cipher

**AIM:** Write a Java/C/C++/Python program to perform encryption and decryption using method of Transposition Technique.

Theory:

**Transposition technique** is an encryption method which is achieved by performing **permutation over the plain text**. Mapping plain text into cipher text using transposition technique is called**transposition cipher**.

## **Transposition Techniques**

1. [Rail Fence Transposition](https://binaryterms.com/transposition-technique-in-cryptography.html#RailFenceTransposition)
2. [Columnar Transposition](https://binaryterms.com/transposition-technique-in-cryptography.html#ColumnarTransposition)
3. [Improved Columnar Transposition](https://binaryterms.com/transposition-technique-in-cryptography.html#ImprovedColumnarTransposition)
4. [Book Cipher/Running Key Cipher](https://binaryterms.com/transposition-technique-in-cryptography.html#BookCipher/RunningKeyCipher)

### **Columnar Transposition Technique**

The columnar transposition cipher is more complex as compared to the rail fence. The steps to obtain cipher text using this technique are as follow:

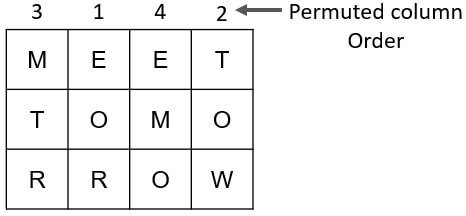
**Step 1:** The plain text is written in the rectangular matrix of the initially defined size in a row by row pattern.

**Step 2:** To obtain the cipher text read the text written in a rectangular matrix column by column. But you have to permute the order of column before reading it column by column. The obtained message is the cipher text message.

To understand the columnar transposition let us take an example:

**Plain text:** meet Tomorrow

Now, put the plain text in the rectangle of a predefined size. For our example, the predefined size of the rectangle would be 3×4. As you can see in the image below the plain text is placed in the rectangle of 3×4. And we have also permuted the order of the column.



Now, to obtain the cipher text we have to read the plain text column by column as the sequence of permuted column order. So, the cipher text obtained by the columnar transposition technique in this example is:

**Cipher Text:** MTREOREMOTOW.

Similar to the rail fence cipher, the columnar cipher can be easily broken. The cryptanalyst only has to try few permutation and combination over the order of column to obtain the permuted order of column and the get the original message. So, a more sophisticated technique was required to strengthen the encryption.

### **Columnar Transposition Technique with Multiple Rounds**

It is similar to the basic columnar technique but is introduced with an improvement. The basic columnar technique is performed over the plain text but more than once. The steps for columnar technique with multiple rounds are as follow:

**Step 1:** The plain text is written in the rectangle of predetermined size row by row.

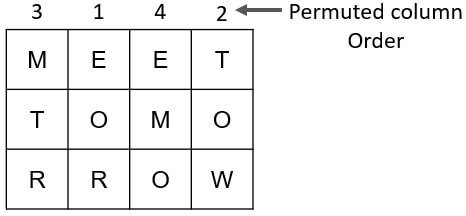
**Step 2:** To obtain the cipher text, read the plain text in the rectangle, column by column. Before reading the text in rectangle column by column, permute the order of columns the same as in basic columnar technique.

**Step 3:** To obtain the final cipher text repeat the steps above multiple time.

Let us discuss one example of a columnar transposition technique for better understanding. We will consider the same example of a basic columnar technique which will help in understanding the complexity of the method:

**Plain Text:** meet Tomorrow

Let us put this plain text in the rectangle of predefined size of 3×4. Proceeding with the next step, the order of the columns of the matrix is permuted as you can see in the image below:



Now after the first round the cipher text obtained is as follow:

**Cipher Text round 1:**  MTREOREMOTOW

Now, again we have to put the cipher text of round 1 in the rectangle of size 3×4 row by row and permute the order of columns before reading the cipher text for round 2. In the second round, the permuted order of the column is 2, 3, 1, 4.

So, the obtained **cipher text for round 2** is MOOTRTREOEMW. In this way, we can perform as many iterations as requires. Increasing the number of iterations increases the complexity of the techniques.

Code:

|  |
| --- |
| # Python3 implementation of  # Columnar 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)))    # This code is contributed by Aditya K |

**Output:**

Encrypted Message: e kefGsGsrekoe\_

Decrypted Message: Geeks for Geeks

Conclusion: Thus we performed encryption and decryption using Transposition technique - **Columnar Transposition**