IS LAB_4

~ Prof. Dr. Aashka Raval

✅ Title 1: Implementation of Columnar Transposition Cipher (Standard)

🎯 Objective:

- To implement the classical Columnar Transposition Cipher for encryption and decryption.
- To understand how plaintext can be rearranged into matrices and read in different orders to form ciphertext.

📘 Introduction:

The Columnar Transposition Cipher is a classical transposition cipher where the plaintext is written row by row into a matrix of fixed column size, based on a key. The ciphertext is generated by reading the matrix column by column in the order defined by the key. This cipher provides a clear understanding of column-based permutations in cryptography.

📚 Concepts Used:

- Symmetric Key Cryptography
- Matrix-based Transposition
- Column Ordering by Key
- Encryption and Decryption via Matrix Rearrangement

🧠 Logic:

Encryption:

1. Remove spaces from plaintext and prepare it for matrix filling.
2. Write plaintext row by row into a matrix of size (rows × key length).
3. Order the columns according to the numerical/lexical order of the key.
4. Read column by column in that order to generate ciphertext.

Decryption:

1. Reconstruct the same matrix dimensions.
2. Place ciphertext characters column by column using the key order.

3. Read row by row to recover the plaintext.

💻 Python Code (Standard Columnar):

```
import math

def encrypt_columnar(message, key):
    message = message.replace(" ", "").upper()
    key_len = len(key)
    rows = math.ceil(len(message) / key_len)
    padded = message.ljust(rows * key_len, 'X')
    matrix = [list(padded[i:i+key_len]) for i in range(0,
len(padded), key_len)]
    print("Matrix:")
    for row in matrix:
        print(row)
    order = sorted(list(enumerate(key)), key=lambda x: x[1])
    ciphertext = ''
    for idx, _ in order:
        for r in matrix:
            ciphertext += r[idx]
    return ciphertext

msg = input("Enter message: ")
key = input("Enter key: ")
cipher = encrypt_columnar(msg, key)
print("Ciphertext:", cipher)
```

🧪 Sample Output:

```
Enter message: I am a Student
Enter key: 1212
Matrix:
['I', 'A', 'M', 'A']
['S', 'T', 'U', 'D']
['E', 'N', 'T', 'X']
Ciphertext: ISEMUTATNADX
```

✅ Conclusion:

The Columnar Transposition Cipher demonstrates the power of simple rearrangements for achieving basic data confidentiality. It lays the foundation for advanced block-based permutation schemes in modern cryptography.

✅ Title 2 (Modified): Implementation of Columnar Cipher with Time-based Rotating Key

---

🎯 Objective:

- To enhance the Columnar Cipher by introducing a dynamic, time-dependent key rotation.
- To increase ciphertext variability even with the same message and base key.

📘 Introduction:

The classical Columnar Cipher uses a static key to decide column ordering. In this modification, the base key is rotated based on the current system time (seconds value). This ensures that the same message encrypted at different times will generate different ciphertexts, thereby strengthening security and making the cipher less predictable.

📚 Concepts Used:

- Symmetric Transposition Cipher
- Time-based Dynamic Key Rotation
- Matrix Construction and Transposition
- Encryption Variability using System Time

🧠 Logic:

Encryption:

1. Take the base key from the user.
2. Get current system time (HH:MM:SS).
3. Use the **seconds value** modulo key length as the rotation shift.
4. Rotate the base key by that shift to create a dynamic key.
5. Fill the plaintext into a matrix row by row.
6. Rearrange columns using the rotated key.
7. Read column by column to generate ciphertext.

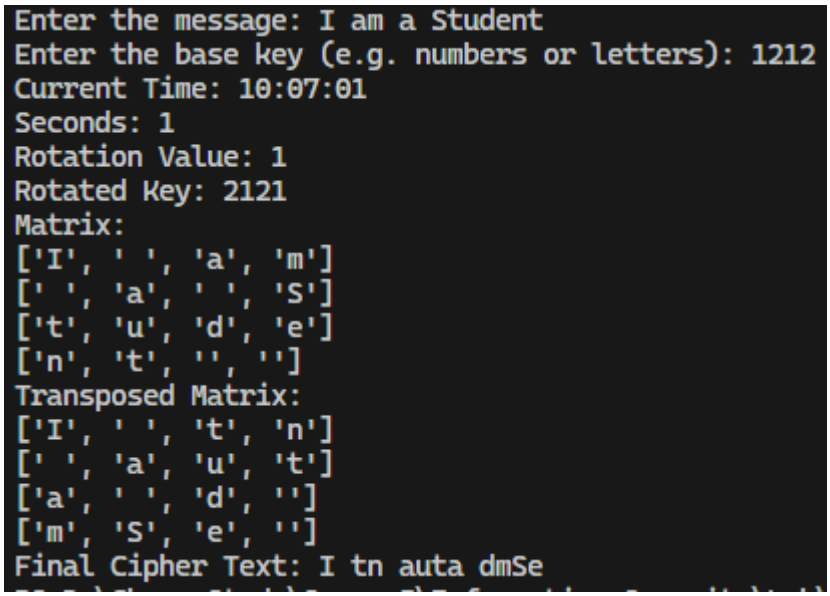
Decryption:

- Requires the same key rotation (time-based shift).
- Rotate the base key identically.
- Place ciphertext into matrix column by column.
- Read row by row to recover plaintext.

💻 Python Code (Time-based Modified Columnar):

```
import math
from datetime import datetime

def create_matrix(message, key_len):
    rows = math.ceil(len(message) / key_len)
    matrix = [['' for _ in range(key_len)] for _ in range(rows)]
    idx = 0
    for i in range(rows):
        for j in range(key_len):
            if idx < len(message):
                matrix[i][j] = message[idx]
                idx += 1
    return matrix

def transpose(matrix):
    return list(map(list, zip(*matrix)))

message = input("Enter the message: ")
base_key = input("Enter the base key (e.g. numbers or letters): ")

now = datetime.now()
seconds = now.second
rotation = seconds % len(base_key)
rotated_key = base_key[rotation:] + base_key[:rotation]

print("Current Time:", now.strftime("%H:%M:%S"))
print("Seconds:", seconds)
print("Rotation Value:", rotation)
print("Rotated Key:", rotated_key)

matrix = create_matrix(message, len(rotated_key))
print("Matrix:")
for row in matrix:
    print(row)

transposed = transpose(matrix)
print("Transposed Matrix:")
for row in transposed:
    print(row)

final = ''.join([''.join(row) for row in transposed])
print("Final Cipher Text:", final)
```

🧪 **Sample Output:**



✅ **Conclusion:**

The Time-based Columnar Cipher enhances the classical design by making the key dynamic and dependent on system time. This ensures ciphertext variability, making the cipher less predictable and more secure against frequency analysis. It demonstrates how simple modifications can significantly increase the strength of traditional cryptographic techniques.
