COMPUTER SECURITY LAB WORK

2023

BRIAN MINJIRE: C026-01-1339/2019

DAVID NDUATI: C026-01-1274/2019

CRYPTOGRAPHY

Columnar Transposition cipher

A Columnar Transposition cipher is a type of transposition cipher, which rearranges the order of the letters in a message to create the ciphertext. The message is written in a grid of rows and columns and then read out column by column in a scrambled order defined by a keyword.

Here's a brief explanation of how it works:

Matrix Formation:

The plaintext message is written into a matrix row by row.

The number of columns in the matrix is determined by the length of the key.

Key Influence:

The key determines the order in which the columns of the matrix are read during encryption.

If the key is "KEY," the columns are read in the order of the letters in the key.

Encryption Process:

The characters are read out column by column, resulting in the encrypted message.

The encrypted message is produced by concatenating the characters column by column according to the order specified by the key.

Example:

Original Message: "HELLO WORLD"

Key: "KEY"

Matrix:

K	**E**	**Y**
H	E	L
L	O	W
O	R	L
D		

Encrypted Message: "HLODEORLWL"

Decryption Process:

The columns are read in the order of the key, but the rows are rearranged back to their original order to retrieve the original message.

The key is essential for both encryption and decryption.

Without the correct key, it is challenging to decipher the message.

Columnar Transposition Ciphers are relatively easy to understand and implement. However, the security of this type of cipher depends on the length and randomness of the key. Longer and more random keys generally make it more secure, but like many classical ciphers, it is susceptible to modern cryptographic attacks and is not suitable for secure communications.

IMPLEMENTATION WITH PYTHON CODE

ENCRYPTION AND DECRYPTION

```
def encrypt(message, key):
    # Remove spaces from the message
    message = message.replace(" ", "")
    # Calculate the number of rows required
    rows = len(message) // len(key)
    if len(message) % len(key) != 0:
        rows += 1
    # Create an empty matrix
    matrix = [['' for _ in range(len(key))] for _ in range(rows)]
    # Populate the matrix with the message
    index = 0
    for i in range(rows):
        for j in range(len(key)):
            if index < len(message):
                matrix[i][j] = message[index]
                index += 1
    # Sort the matrix columns based on the key
```

```
    sorted_matrix = sorted(matrix, key=lambda x: [key.index(c) for c in key])
    # Read the encrypted message from the sorted matrix
    encrypted_message = ''
    for j in range(len(key)):
        for i in range(rows):
            encrypted_message += sorted_matrix[i][j]
    return encrypted_message
# Example usage:
message = input("Enter message to encript: ")
#TO DETERMINE THE LENGTH OF THE KEY FOR ENCRIPTION
key = "KEY"
encrypted_message = encrypt(message, key)
print("Original Message:", message)
print("Encrypted Message:", encrypted_message)

#TO DECRYPT THE ENCRYPTED MESSAGE RUN THE FOLLOWING
def decrypt(encrypted_message, key):
    # Calculate the number of rows required
    rows = len(encrypted_message) // len(key)
    if len(encrypted_message) % len(key) != 0:
        rows += 1
    # Create an empty matrix
    matrix = [['' for _ in range(len(key))] for _ in range(rows)]
    # Calculate the number of empty cells in the last row
    empty_cells = len(key) - (len(encrypted_message) % len(key))
    # Populate the matrix with the encrypted message
    index = 0
    for j in range(len(key)):
        for i in range(rows):
            if i == rows - 1 and j >= len(key) - empty_cells:
```

```
            continue  # Skip the empty cells in the last row
        matrix[i][j] = encrypted_message[index]
        index += 1
    # Sort the matrix columns based on the key
    sorted_matrix = sorted(matrix, key=lambda x: [key.index(c) for c in key], reverse=True)
    # Read the decrypted message from the sorted matrix
    decrypted_message = ''
    for i in range(rows):
        for j in range(len(key)):
            decrypted_message += sorted_matrix[i][j]
    return decrypted_message
# Example usage:
decrypted_message = decrypt(encrypted_message, key)
print("Decrypted Message:", decrypted_message)
```

OUTPUT

```
============== RESTART: C:/Users/USER/columnar transposition.py =============
Enter message to encript: GOOD DAY
Original Message: GOOD DAY
Encrypted Message: GDYODOA
Decrypted Message: GOODDAY
```