Mohammad Sufyan Azam (2020312),
Aamleen Ahmad (2020002)

Assignment 1

We had to implement Project 2, i.e., transposition cipher algorithm on {a-z}.

Introduction

Our Transposition Cipher Program is a python script designed to perform encryption, decryption, and a brute-force attack using a transposition cipher. The transposition cipher is a method of encryption where the positions of characters in the plaintext are shifted according to a specified key.

The program allows users to encrypt and decrypt messages using a transposition cipher, as well as attempt to discover the key through a brute-force attack. The encryption process involves generating a hash value of the plaintext, creating a transposition matrix, and rearranging the characters according to the key on a high level overview. Decryption reverses this process, and the brute-force attack attempts to find the key by systematically trying all possible permutations with all possible key length combinations where key length ranges from 3 to 9. It can also take a key deciphered from a previous cipher as an input and try to open the next cipher with it. If it is unsuccessful, then it will try the permutations of the given key and then advance to (key_length+1)%10, thus trying out all possible permutations of all possible key lengths.

Usage

Encrypting Messages

To encrypt messages, users can provide a plaintext file as input that may or may not contain multiple plaintexts, each plaintext being in a separate line. The program then generates a random key for encryption and then uses the key to create a transposition matrix, and produces the encrypted text. The encrypted text and the corresponding key are then saved to separate files called cipher_text.txt and keys.txt correspondingly.
This is done so that the cipher_text can be sent over an unsecure channel while the keys can be sent over a secure channel to the receiver, thus disallowing any intruder to get access to the keys along with the cipher text.

Mohammad Sufyan Azam (2020312),
Aamleen Ahmad (2020002)

Code -

```
def encrypt(plain_text, key):
     '''Calculates the hash value of the new_plain_text and appends it to
the new_plain_text.
    Encrypts the message using the key and returns the cipher_text
         to_be_encrypted = plain_text + garbage_characters + "#" +
hash_value'''

    # key = generate_key()
    new_plain_text = generate_new_plain_text(plain_text, len(key))
    hash_value = generate_hash_value(new_plain_text)

    print('New plain text: ' + new_plain_text)
    print('Hash value: ' + hash_value)
    print('Key: ' + key)

    to_be_encrypted = new_plain_text + "#" + hash_value
    cipher_text = ""

    print('Message to be encrypted: ' + to_be_encrypted)

    matrix = transposition_matrix(to_be_encrypted, key)
    # print("Transposition matrix: ")
    # for row in matrix:
    #     print(row)

    ordered_indices = order_matrix_indices(key)

    for i in range(len(key)):
        for j in range(len(matrix)):
            cipher_text += matrix[j][ordered_indices[i][1]]


    return cipher_text
```

Mohammad Sufyan Azam (2020312),
Aamleen Ahmad (2020002)

Workflow

This function takes the help of two other functions, transposition matrix, and the order_matrix_indices to generate the encrypted text. The transposition matrix function, for encryption, creates a matrix by filling rows with characters from the new_plain_text and the columns being equal to the key length. The order_matrix_indices() determines the order of columns for a matrix based on the specified key. It returns a list of tuples, where each tuple contains the original index and the new index of a column in ascending order according to the key.

Decrypting Messages

Decryption requires the input of the cipher_text.txt file and the corresponding keys.txt file to decrypt the messages. The program reads these files, performs the decryption process, and stores the decrypted text along with the hash value in a separate file called decrypted_text.txt.

Code -

```python
def decrypt(cipher_text, key):
        '''Decrypts the cipher_text using the key and returns the
decrypted_text.
    Removes the hash value from the decrypted_text and returns it.'''

    decrypted_text = ""
    matrix = transposition_matrix(cipher_text, key, encrypt=False)

    # Flatten the transposition matrix
    decrypted_text = ''.join(matrix.flatten())
    # print(decrypted_text)

    split_index = decrypted_text.rfind("#")
    hash_value = decrypted_text[split_index+1:]
    decrypted_text = decrypted_text[:split_index]

    valid = verify_decrypted_text(decrypted_text, hash_value)

    return decrypted_text, hash_value, valid
```

Workflow -

It takes the help of transposition_matrix() and verify_decrypted_text() function to compute the decrypted text. The verify_decrypted_text() function computes the hash value of the decrypted text and compares it with the hash value that was received. If they both match then it returns true else false.

Brute Force Attack

The brute-force attack option allows users to attempt to discover the key used for encryption. The program reads a cipher text file, systematically tries all possible keys, starting from key length 3 and goes until key length 9 and checks for valid decryption. If successful, the key, decrypted text, and hash value are displayed and the keys are stored in a separate file called brute_force_attack.txt. We can also specify a key to start the brute force attack from that key.
Note: *The code for the brute force attack is in the main file (2020312_2020002.py) which is not posted here due to the document length constraints.*

Workflow

The _apply_visual_effects() function applies the visual effects for decrypting the key through the brute force attack. It displays a flashing message indicating successful hacking and prompts the user to press Enter to reveal details. It also uses a permutation function that finds the next greater permutation of the key.
The performance of the brute-force attack depends on the length of the cipher text and the key space. Longer keys will require more time to try all possible permutations. Users should be cautious when applying this method to large cipher texts or when the key space is extensive.