**Practical No: 3**

**Practical Name: Study and Implement a program for Rail Fence Cipher.**

**Roll No: 22BCP250**

**Name: Anusi Patel**

**Division and batch: Div4 G7**

**Original Approach:**

**Introduction:** The Rail Fence Cipher is a classic example of a transposition cipher, where the letters of the plaintext are rearranged to form the ciphertext. It gets its name from the way the encryption process resembles the zigzag pattern of rails on a fence. Unlike substitution ciphers, which replace letters with others, the Rail Fence Cipher merely changes the order of the letters, making it a simple yet effective method for obscuring a message.

**How the Rail Fence Cipher Works:**

1. **Preparation**:

* The key to the Rail Fence Cipher is the number of "rails" (or rows) used. This number determines how many levels the message will be split into.
* For example, if you choose 3 rails, the message will be split into three different levels or lines.

1. **Encryption Process**:

* **Writing the Message**:
  + - * The plaintext message is written in a zigzag pattern across the chosen number of rails.
      * Starting from the top rail, the letters are placed diagonally downwards until the bottom rail is reached, after which the direction is reversed to move back up to the top rail.
* **Reading the Message**:
  + - * Once the entire message is written in this zigzag pattern, the ciphertext is created by reading the characters from each rail sequentially from top to bottom.

1. **Decryption**:

* To decrypt a Rail Fence Cipher, you need to reverse the process:
  + - The number of characters that fall on each rail must be determined based on the length of the ciphertext and the number of rails.
    - The ciphertext is then rewritten back into the zigzag pattern.
    - Finally, the plaintext is reconstructed by reading the characters in the order they were originally written.

**Example:**

Let's use a 3-rail cipher to encrypt the phrase "ATTACK AT DAWN".

* **Step 1: Write the Message in Zigzag Form**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | . | . | . | C | . | . | . | D | . | . | . |
| . | T | . | A | . | K | . | T | . | A | . | N |
| . | . | T | . | . | . | A | . | . | . | W | . |

* **Step 2: Read Each Line Sequentially**
  + Top Rail: A C D
  + Middle Rail: T A K T A N
  + Bottom Rail: T A W
  + Combined Ciphertext: ACDTAKTANTAW

**Steps to Decrypt the Rail Fence Cipher:**

1. **Determine the Rail Pattern**:
   * Given the length of the ciphertext and the number of rails, distribute the characters evenly (or as close as possible) among the rails.
2. **Reconstruct the Zigzag Pattern**:
   * Write the characters back into the zigzag pattern as they would have been during encryption.
3. **Reconstruct the Plaintext**:
   * Once the pattern is filled in, read the characters in the order they were written to retrieve the original message.

**Source Code:**

alphabet\_lower = "abcdefghijklmnopqrstuvwxyz"

alphabet\_upper = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

def preprocess\_text(plaintext):

processed\_text = ''

for char in plaintext:

if char != ' ':

if char in alphabet\_lower:

index = 0

for i in range(len(alphabet\_lower)):

if alphabet\_lower[i] == char:

index = i

break

processed\_text += alphabet\_upper[index]

else:

processed\_text += char

return processed\_text

def rail\_fence\_encrypt(plaintext, key):

matrix = [['' for \_ in range(len(plaintext))] for \_ in range(key)]

row = 0

direction = 1

for i in range(len(plaintext)):

matrix[row][i] = plaintext[i]

row += direction

if row == 0 or row == key - 1:

direction \*= -1

ciphertext = ''

for r in range(key):

for c in range(len(plaintext)):

if matrix[r][c] != '':

ciphertext += matrix[r][c]

return ciphertext

def rail\_fence\_decrypt(ciphertext, key):

matrix = [['' for \_ in range(len(ciphertext))] for \_ in range(key)]

row = 0

direction = 1

index = 0

for i in range(len(ciphertext)):

matrix[row][i] = '\*'

row += direction

if row == 0 or row == key - 1:

direction \*= -1

for r in range(key):

for c in range(len(ciphertext)):

if matrix[r][c] == '\*':

matrix[r][c] = ciphertext[index]

index += 1

plaintext = ''

row = 0

direction = 1

for i in range(len(ciphertext)):

plaintext += matrix[row][i]

row += direction

if row == 0 or row == key - 1:

direction \*= -1

return plaintext

plaintext = input("Enter the plaintext: ")

key = 3

processed\_plaintext = preprocess\_text(plaintext)

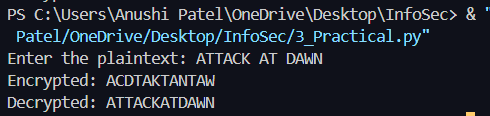
encrypted = rail\_fence\_encrypt(processed\_plaintext, key)

print(f"Encrypted: {encrypted}")

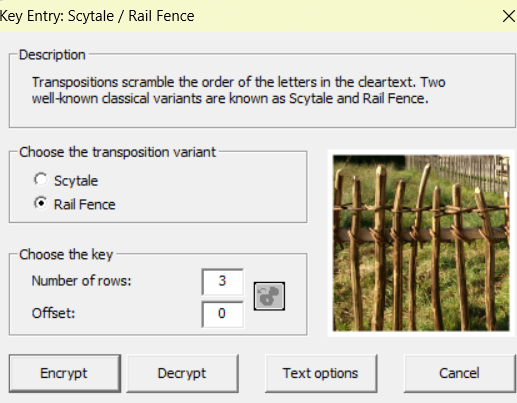
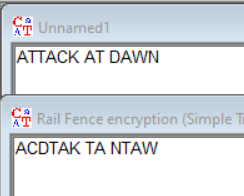
decrypted = rail\_fence\_decrypt(encrypted, key)

print(f"Decrypted: {decrypted}")

**Output:**

****

**Cryptool:**

**** ****

**Revised Approach:**

**Introduction:** The modified version of the Rail Fence Cipher implemented here combines the Rail Fence Cipher and the Caesar Cipher to enhance security and make the encryption more resistant to attacks. This approach utilizes a three-step process: applying the Rail Fence Cipher, followed by the Caesar Cipher, and then applying the Rail Fence Cipher again. This layered encryption adds complexity, making it more challenging to decipher the ciphertext without knowing all the keys involved.

**Drawbacks of the Simple Rail Fence Cipher:**

1. **Predictable Pattern**: The Rail Fence Cipher creates a recognizable zigzag pattern, which attackers familiar with the technique can exploit. Once the key (number of rails) is determined, decrypting the message becomes straightforward.
2. **Limited Security**: The basic Rail Fence Cipher is a transposition cipher that only rearranges the letters, without altering their identities. This makes it vulnerable to pattern recognition and frequency analysis.
3. **Key Limitation**: The effectiveness of the Rail Fence Cipher is directly tied to the key size (number of rails). A small key size limits the security, while a large key size can make the ciphertext look more scrambled but still susceptible to certain cryptanalytic methods.

**How This Modified Version Improves Security:**

1. **Dual Rail Fence Layers**:

* **Improvement**: By applying the Rail Fence Cipher twice—once before and once after the Caesar Cipher—this method introduces additional complexity. The first application scrambles the plaintext, the Caesar Cipher shifts the characters, and the second Rail Fence Cipher further scrambles the result. This dual-layered approach makes the pattern less recognizable and more challenging to reverse-engineer.

1. **Incorporation of Caesar Cipher**:

* **Improvement**: The Caesar Cipher adds a substitution layer, altering the letters rather than just their order. This combination of transposition (Rail Fence) and substitution (Caesar Cipher) enhances the overall security, making applying traditional frequency analysis and pattern recognition techniques harder.

1. **Increased Keyspace**:

* **Improvement**: The modified method uses three keys: one for each Rail Fence Cipher and one for the Caesar Cipher. This increased keyspace significantly raises the difficulty of brute-force attacks, as an attacker would need to determine all three keys to decrypt the message.

**Example:**

* **Plaintext**: "ATTACK AT DAWN"
* **Keys**:
  + **Rail Fence Key 1**: 3
  + **Caesar Cipher Key**: 4
  + **Rail Fence Key 2**: 3

**Step 1: Preprocess Text**

* **Remove spaces and convert to uppercase**: "ATTACKATDAWN"

**Step 2: First Rail Fence Cipher (Key = 3)**

**Rail Fence Pattern (3 rails)**:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | . | . | . | C | . | . | . | D | . | . | . |
| . | T | . | A | . | K | . | T | . | A | . | N |
| . | . | T | . | . | . | A | . | . | . | W | . |

To fill the rails, follow the zigzag pattern:

1. **Rail 1**: Start with "A", move to "C", then "D", and place the rest in this rail as you zigzag through the text.
2. **Rail 2**: Place remaining characters in this rail as you continue zigzagging.
3. **Rail 3**: Place the rest of the characters in this rail.

* **Rail 1**: "A C D"
* **Rail 2**: "T A K T A N"
* **Rail 3**: "T A W"

**Ciphertext after Rail Fence Encryption**: Combine the characters from each rail:

* **Rail 1**: "ACD"
* **Rail 2**: "TAKTAN"
* **Rail 3**: "TAW"

**Final Ciphertext**: "ACDTAKTANTAW"

**3. Caesar Cipher (Shift by 4)**

**Shift each letter by 4 positions in the alphabet**:

* A → E
* C → G
* D → H
* T → X
* K → O
* A → E
* N → R
* W → A

**Ciphertext after Caesar Cipher**:

* **Rail 1**: "EGH"
* **Rail 2**: "XEOXER"
* **Rail 3**: "XEA"

Combine these:

**Ciphertext**: "EGHXEOXERXEA"

**4. Second Rail Fence Cipher (Key = 3)**

**Rail Fence Pattern (3 rails)**:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| E | . | . | . | E | . | . | . | R | . | . | . |
| . | G | . | X | . | O | . | E | . | X | . | A |
| . | . | H | . | . | . | X | . | . | . | E | . |

To fill the rails, follow the zigzag pattern again:

1. **Rail 1**: Place characters as you zigzag through the encrypted text.
2. **Rail 2**: Place remaining characters in this rail.
3. **Rail 3**: Place the rest of the characters in this rail.

* **Rail 1**: "EER"
* **Rail 2**: "GXOEXA"
* **Rail 3**: "HXE"

**Final Ciphertext**: Combine the characters from each rail:

**Combined Ciphertext**: "EERGXOEXAHXE"

**Source Code:**

alphabet\_upper = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

alphabet\_lower = "abcdefghijklmnopqrstuvwxyz"

def preprocess\_text(plaintext):

processed\_text = ''

for char in plaintext:

if char != ' ':

if 'a' <= char <= 'z':

for i in range(len(alphabet\_lower)):

if alphabet\_lower[i] == char:

processed\_text += alphabet\_upper[i]

break

else:

processed\_text += char

return processed\_text

def rail\_fence\_encrypt(plaintext, key):

matrix = [['' for \_ in range(len(plaintext))] for \_ in range(key)]

row = 0

direction = 1

for i in range(len(plaintext)):

matrix[row][i] = plaintext[i]

row += direction

if row == 0 or row == key - 1:

direction \*= -1

ciphertext = ''

for r in range(key):

for c in range(len(plaintext)):

if matrix[r][c] != '':

ciphertext += matrix[r][c]

return ciphertext

def rail\_fence\_decrypt(ciphertext, key):

matrix = [['' for \_ in range(len(ciphertext))] for \_ in range(key)]

row = 0

direction = 1

for i in range(len(ciphertext)):

matrix[row][i] = '\*'

row += direction

if row == 0 or row == key - 1:

direction \*= -1

index = 0

for r in range(key):

for c in range(len(ciphertext)):

if matrix[r][c] == '\*':

matrix[r][c] = ciphertext[index]

index += 1

plaintext = ''

row = 0

direction = 1

for i in range(len(ciphertext)):

plaintext += matrix[row][i]

row += direction

if row == 0 or row == key - 1:

direction \*= -1

return plaintext

def caesar\_encrypt(text, key):

result = ""

for char in text:

if 'A' <= char <= 'Z':

for i in range(len(alphabet\_upper)):

if alphabet\_upper[i] == char:

new\_position = (i + key) % 26

result += alphabet\_upper[new\_position]

break

return result

def caesar\_decrypt(text, key):

result = ""

for char in text:

if 'A' <= char <= 'Z':

for i in range(len(alphabet\_upper)):

if alphabet\_upper[i] == char:

new\_position = (i - key + 26) % 26

result += alphabet\_upper[new\_position]

break

return result

def modified\_cipher\_encrypt(plaintext, rail\_key1, caesar\_key, rail\_key2):

processed\_plaintext = preprocess\_text(plaintext)

step1 = rail\_fence\_encrypt(processed\_plaintext, rail\_key1)

step2 = caesar\_encrypt(step1, caesar\_key)

final\_encryption = rail\_fence\_encrypt(step2, rail\_key2)

return final\_encryption

def modified\_cipher\_decrypt(ciphertext, rail\_key1, caesar\_key, rail\_key2):

step1 = rail\_fence\_decrypt(ciphertext, rail\_key2)

step2 = caesar\_decrypt(step1, caesar\_key)

final\_decryption = rail\_fence\_decrypt(step2, rail\_key1)

return final\_decryption

plaintext = input("Enter the plaintext: ")

rail\_key1 = 3

rail\_key2 = 3

caesar\_key = 4

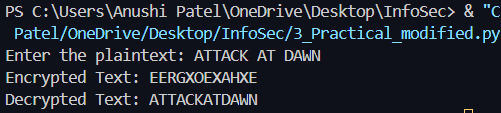
encrypted\_text = modified\_cipher\_encrypt(plaintext, rail\_key1, caesar\_key, rail\_key2)

print(f"Encrypted Text: {encrypted\_text}")

decrypted\_text = modified\_cipher\_decrypt(encrypted\_text, rail\_key1, caesar\_key, rail\_key2)

print(f"Decrypted Text: {decrypted\_text}")

**Output:**

****

**Cryptanalysis on Rail Fence Cipher:**

1. **Frequency Analysis**:

* **Description**: This technique involves analyzing the frequency of letters or groups of letters in the ciphertext.
* **Application**: Although the Rail Fence Cipher does not substitute characters, analyzing the frequency of characters in the ciphertext and comparing it to the frequency of letters in the language can give clues about the pattern used in the cipher.

1. **Brute Force Attack**:

* **Description**: Trying all possible keys (numbers of rails) to find the one that correctly decrypts the ciphertext.
* **Application**: Since the Rail Fence Cipher is a transposition cipher with a small keyspace (number of rails), a brute force attack is practical. If the number of rails is small, all possible configurations can be tried quickly.

1. **Pattern Recognition**:

* **Description**: Identifying patterns in the ciphertext that correspond to common phrases or structures in the plaintext.
* **Application**: The zigzag pattern used in Rail Fence Cipher can sometimes be deduced by examining the ciphertext, especially if the plaintext contains predictable words or phrases.

**Conclusion:**

The modified Rail Fence Cipher significantly enhances the security and complexity of the traditional encryption method by integrating dual layers of encryption. This improved version applies the Rail Fence Cipher twice, with a Caesar Cipher applied in between, thereby adding substantial complexity to the encryption process. The dual application of the Rail Fence Cipher, combined with the substitution provided by the Caesar Cipher, makes the encryption more resistant to pattern recognition and frequency analysis—two common weaknesses in the original method. Additionally, the careful handling of text, including the pre-processing steps to manage spaces and letter case, ensures the integrity and accuracy of both encryption and decryption. Overall, this modification addresses the primary vulnerabilities of the traditional Rail Fence Cipher, offering a more secure and sophisticated approach that is better equipped to withstand modern cryptographic attacks.

**References:**

1. [IEEE Xplore Full-Text PDF(A New Algorithm Combining Substitution & Transposition Cipher Techniques for Secure Communication):](https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8300777) (Umang Bhargava, Aparna Sharma, Raghav Chawla, Prateek Thakral From National Institute of Technology Kurukshetra, India)
2. [IJTRD207.pdf(Modified Ceaser Cipher and Rail fence Technique to Enhance Security):](https://www.ijtrd.com/papers/IJTRD207.pdf) (Baljit Saini, Lecturer, Computer Department, K.D.Polytechnic, Patan, Gujrat, India)