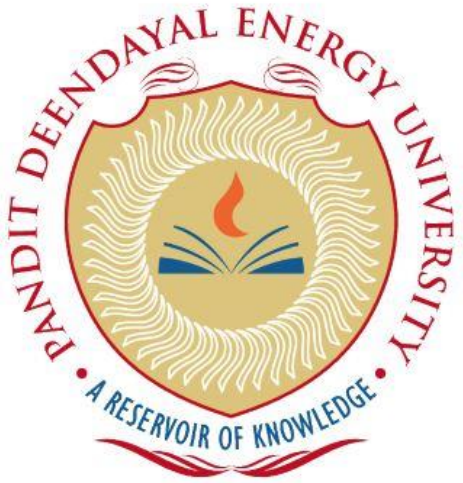
PANDIT DEENDAYAL ENERGY UNIVERSITY

SCHOOL OF TECHNOLOGY



Course: Information Security

Course Code: 20CP304P

LAB MANUAL

B.Tech. (Computer Engineering)

Semester 5

Submitted To:

Dr. Hargeet Kaur

Submitted By:

Aryan Randeriya

22BCP469D

G3

INDEX

S. No.	List of experiments	Date	Sign
1	Study and Implement a program for Caesar Cipher		
2	Study and Implement a program for 5x5 Playfair Cipher to encrypt and decrypt the message.		
3	Study and Implement a program for Rail Fence Cipher with columnar transposition		
4	Study and implement a program for columnar Transposition Cipher		
5	Study and implement a program for Vigenère Cipher		
6	Study and Implement a program for n-gram Hill Cipher		
7	Study and Use of RSA algorithm (encryption and decryption)		
8	Study and implement a program of the Digital Signature with RSA algorithm (Reverse RSA)		
9	Study and Use of Diffie-Hellman Key Exchange		
10	Design cipher with detailed explanation. Sample as follows. a) Write a program to encrypt the plaintext with the given key. E.g. plaintext GRONSFELD with the key 1234. Add 1 to G to get H (the letter 1 rank after G is H in the alphabet), then add 2 to C or E (the letter 2 ranks after C is E), and so on. Use smallest letter from plaintext as filler. b) Encrypt the input words PLAINTEXT= RAG BABY to obtain CIPHERTEXT = SCJ DDFD		

Experiment 3

AIM: Study and Implement a program for Rail Fence Cipher

Introduction:

The Rail Fence Cipher is a transposition cipher that encrypts a message by writing it in a zigzag pattern across multiple lines and then reading it off line by line. Here's how it works:

Encryption

1. Write the plaintext in a zigzag pattern across a specified number of "rails" (rows).
2. Read the message row by row to get the ciphertext.

Decryption

1. Reconstruct the zigzag pattern using the ciphertext and the number of rails.
2. Read the plaintext by following the zigzag pattern.

Program:

```
def user_input() -> tuple:
    while True:
        try:
            # Prompt user for the key and convert it to an integer
            keystring: int = int(
                input("Enter a numeric key (number of rails): "))
            # Prompt user for the plaintext and remove spaces
            plaintext: str = input(
                "Enter a sentence to encrypt: ").replace(" ", "")
            # Check if the key is a positive integer
            if keystring <= 0:
                raise ValueError("Key must be a positive integer.")
            return keystring, plaintext
        except ValueError as e:
            # Display error message and prompt user to try again if input is
invalid
            print(f"Error: {e}")
            print("Please try again.")


def encrypt(key: int, text: str) -> str:
    """Encrypt the given text using the Rail Fence Cipher with the specified
key."""
    if key == 1:
        # If key is 1, no encryption is needed (only one rail)
```

```
        return text

    # Initialize a list of empty strings for each rail
    rails = ['' for _ in range(key)]
    rail = 0  # Start on the first rail
    direction = 1  # Move direction: 1 for down, -1 for up

    # Iterate over each character in the text
    for index, char in enumerate(text):
        # Append character to the current rail
        rails[rail] += char
        # Move to the next rail based on the direction
        rail += direction
        # Reverse direction if reaching the top or bottom rail
        if rail == 0 or rail == key - 1:
            direction *= -1

    # Concatenate all rails to form the encrypted text
    encrypted_text = ''.join(rails)
    return encrypted_text


def main():
    """Main function to handle user input and perform encryption."""
    keystring, plaintext = user_input()

    # Check if the key is larger than the length of the plaintext
    if keystring > len(plaintext):
        print("Key is larger than the length of the plaintext.")
        keystring = len(plaintext)  # Adjust key to the length of the
plaintext
        ciphertext = plaintext  # No encryption needed if key is too large
    else:
        # Encrypt the plaintext using the Rail Fence Cipher
        ciphertext = encrypt(keystring, plaintext)

    # Display the encrypted text
    print(f"\nEncrypted text: {ciphertext}")


if __name__ == "__main__":
    main()
```

Output:

```
Enter a numeric key (number of rails): 3
Enter 'e' to encrypt or 'd' to decrypt: e
Enter the text to encrypt or decrypt: aryan randeriya

Encrypted text: andyrarneiayar
```

```
Enter a numeric key (number of rails): 3
Enter 'e' to encrypt or 'd' to decrypt: d
Enter the text to encrypt or decrypt: andyrarneiayar

Decrypted text: aryanranderiya
```

Revised Approach

The revised approach enhances the Rail Fence Cipher implementation by adding support for multiple rounds of encryption or decryption, as specified by the user. Unlike the basic version, which handles a single round of encryption or decryption, this version allows users to repeatedly apply the cipher, offering increased security.

Code:

```
def user_input() -> tuple:
    """Get user input for the action (encrypt/decrypt), key, text, and number
of rounds."""
    while True:
        try:
            action = input(
                "Would you like to encrypt or decrypt? (e/d): ").lower()
            if action not in ['e', 'd']:
                raise ValueError(
                    "Please enter 'e' for encryption or 'd' for decryption.")

            keystring: int = int(
                input("Enter a numeric key (number of rails): "))
            text: str = input("Enter the text: ").replace(" ", "")
            rounds: int = int(input("Enter the number of rounds: "))
            if keystring <= 0 or rounds <= 0:
                raise ValueError("Key and rounds must be positive integers.")

            return action, keystring, text, rounds
        except ValueError as e:
            print(f"Error: {e}")
            print("Please try again.")


def encrypt(key: int, text: str) -> str:
    """Encrypt the given text using the Rail Fence Cipher with the specified
key."""
    if key == 1:
        return text

    rails = ['' for _ in range(key)]
    rail = 0
    direction = 1

    for index, char in enumerate(text):
        rails[rail] += char
        rail += direction
        if rail == 0 or rail == key - 1:
            direction *= -1
```

```
    encrypted_text = ''.join(rails)
    return encrypted_text


def decrypt(key: int, text: str) -> str:
    """Decrypt the given text using the Rail Fence Cipher with the specified
key."""
    if key == 1:
        return text

    # Determine the length of each rail
    rail_lengths = [0] * key
    rail = 0
    direction = 1
    for index in range(len(text)):
        rail_lengths[rail] += 1
        rail += direction
        if rail == 0 or rail == key - 1:
            direction *= -1

    # Reconstruct the rails
    rails = []
    index = 0
    for length in rail_lengths:
        rails.append(text[index:index + length])
        index += length

    # Reconstruct the original text
    decrypted_text = ''
    rail = 0
    direction = 1
    for index in range(len(text)):
        decrypted_text += rails[rail][0]
        rails[rail] = rails[rail][1:]
        rail += direction
        if rail == 0 or rail == key - 1:
            direction *= -1

    return decrypted_text


def main():
    action, keystring, text, rounds = user_input()

    if action == 'e':
        result = text
        for _ in range(rounds):
            result = encrypt(keystring, result)
        print(f"\nEncrypted text after {rounds} rounds: {result}")
```

```
    elif action == 'd':
        result = text
        for _ in range(rounds):
            result = decrypt(keystring, result)
        print(f"\nDecrypted text after {rounds} rounds: {result}")


if __name__ == "__main__":
    main()
```

Output:

```
Would you like to encrypt or decrypt? (e/d): e
Enter a numeric key (number of rails): 3
Enter the text: aryan randeriya
Enter the number of rounds: 3

Encrypted text after 3 rounds: anirraynydaear
```

```
Would you like to encrypt or decrypt? (e/d): d
Enter a numeric key (number of rails): 3
Enter the text: anirraynydaear
Enter the number of rounds: 3

Decrypted text after 3 rounds: aryanranderiya
```