Lab 3: Rail Fence Cipher:

Introduction:

The Rail Fence Cipher is a type of transposition cipher in which plaintext is written diagonally in a zigzag pattern across multiple rows (rails) and then read row by row to get the encrypted text. The number of rows used is referred to as the depth.

How It Works:

1. Choose the number of rows (rails) for the cipher.
2. Write the plaintext in a zigzag pattern across the rows.
3. Read the characters row-wise to obtain the ciphertext.
4. To decrypt, reconstruct the zigzag pattern based on the number of rails and read the text row by row in the original order.

Example:

Plaintext: "HELLO WORLD"

Rails: 2

Step 1: Write in Zigzag Pattern

H L O O L

E L W R D

Step 2: Read row-wise to get the ciphertext:

Ciphertext: "HLOOL ELWRD"

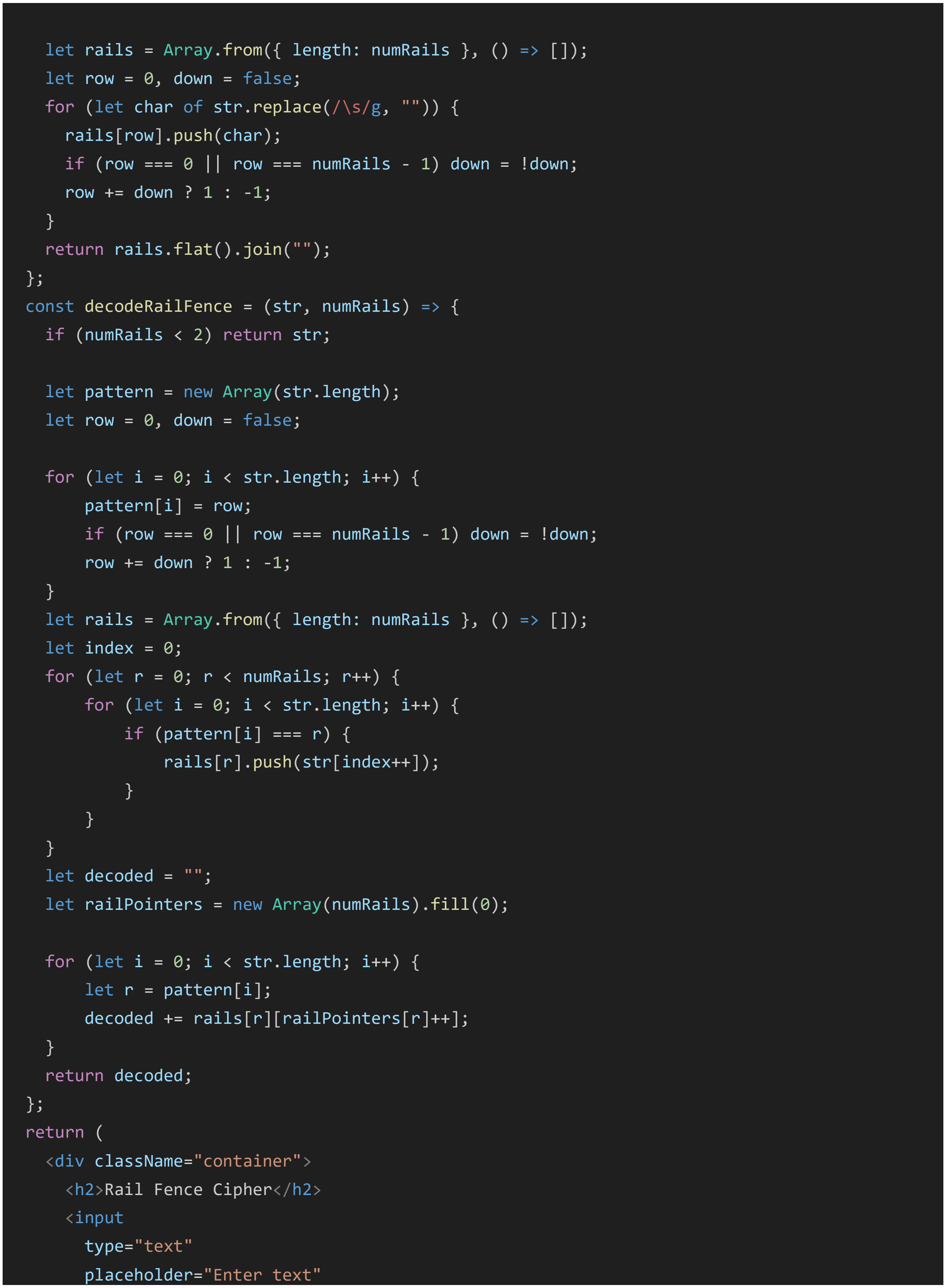
Decryption Process:

To decrypt, the zigzag pattern is reconstructed using the number of rails, and then characters are placed accordingly to reveal the original message.

Implementation in React:

Code:

```
import { useState } from "react";
import "./RailFenceCipher.css";
const RailFenceCipher = () => {
  const [text, setText] = useState("");
  const [depth, setDepth] = useState(2);
  const [result, setResult] = useState("");

  const encodeRailFence = (str, numRails) => {
    if (numRails < 2) return str;
```

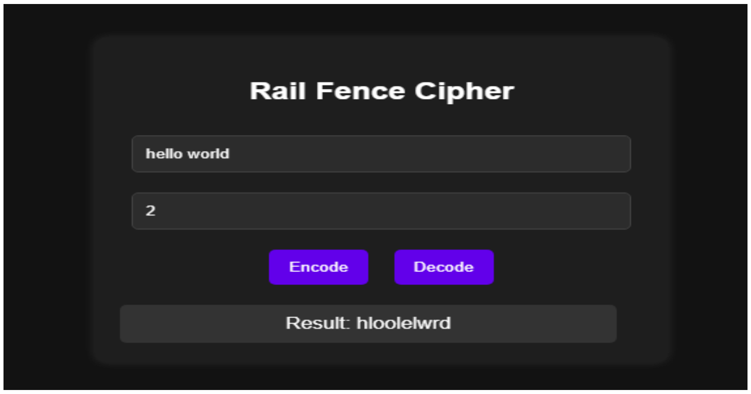


```
  let rails = Array.from({ length: numRails }, () => []);
  let row = 0, down = false;
  for (let char of str.replace(/\s/g, "")) {
    rails[row].push(char);
    if (row === 0 || row === numRails - 1) down = !down;
    row += down ? 1 : -1;
  }
  return rails.flat().join("");
};
const decodeRailFence = (str, numRails) => {
  if (numRails < 2) return str;

  let pattern = new Array(str.length);
  let row = 0, down = false;

  for (let i = 0; i < str.length; i++) {
      pattern[i] = row;
      if (row === 0 || row === numRails - 1) down = !down;
      row += down ? 1 : -1;
  }
  let rails = Array.from({ length: numRails }, () => []);
  let index = 0;
  for (let r = 0; r < numRails; r++) {
      for (let i = 0; i < str.length; i++) {
          if (pattern[i] === r) {
              rails[r].push(str[index++]);
          }
      }
  }
  let decoded = "";
  let railPointers = new Array(numRails).fill(0);

  for (let i = 0; i < str.length; i++) {
      let r = pattern[i];
      decoded += rails[r][railPointers[r]++];
  }
  return decoded;
};
return (
  <div className="container">
    <h2>Rail Fence Cipher</h2>
    <input
      type="text"
      placeholder="Enter text"
```

```
        value={text}
        onChange={(e) => setText(e.target.value)}/>
      <input
        type="number"
        placeholder="Depth"
        value={depth}
        min={2}
        onChange={(e) => setDepth(Math.max(2, Number(e.target.value)))}/>
      <div className="button-group">
        <button onClick={() => setResult(encodeRailFence(text, depth))}>Encode</button>
        <button onClick={() => setResult(decodeRailFence(text, depth))}>Decode</button>
      </div>
      <div className="result-box">Result: {result}</div>
    </div>
  );
};
export default RailFenceCipher;
```

Output:

Encode:



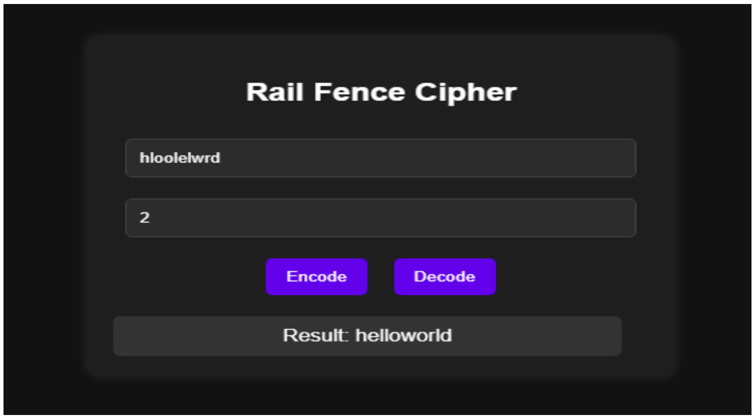
Rail Fence Cipher

hello world

2

Result: hloolelwrd

Decode:



Rail Fence Cipher

hloolelwrd

2

Result: helloworld

Advantages:

- Simple to implement and understand.
- Provides basic security against casual decryption.

Disadvantages:

- Vulnerable to frequency analysis and brute-force attacks due to limited security.
- Not suitable for modern cryptographic security needs.

The Rail Fence Cipher is mainly used for educational purposes and historical cryptography studies rather than practical encryption today.