Lab 4: Columnar Transposition Cipher:

Introduction

The Columnar Transposition Cipher is a classical encryption technique that secures messages by rearranging their letters based on a chosen keyword. This method solely changes the order of the letters, preserving the original characters but making the text unintelligible without the correct key.

How It Works

The cipher works by writing the plaintext message in a grid (rows and columns) based on the length of the keyword and then rearranging the columns according to the alphabetical order of the keyword.

Encryption Process

1. **Choose a Key:** Select a keyword (e.g., "SECRET").
2. **Write the Plaintext in a Grid:** Write the plaintext row by row in a table with as many columns as the length of the keyword.
3. **Sort Columns Based on Key:** Arrange the columns in alphabetical order of the key's letters.
4. **Read Column-wise:** Read the columns in the new order to get the ciphertext.

Example

- **Plaintext:** "HELLO WORLD"
- **Key:** "KEY"

Writing it in a grid (assuming padding with '_'):

K E Y

H E L

L O W

O R L

D _ _

Sorting columns based on key (E comes first, then K, then Y):

E K Y

E H L

O L W

R O L

_ D _

Ciphertext: "EOR_HLODLWL_"

Decryption Process

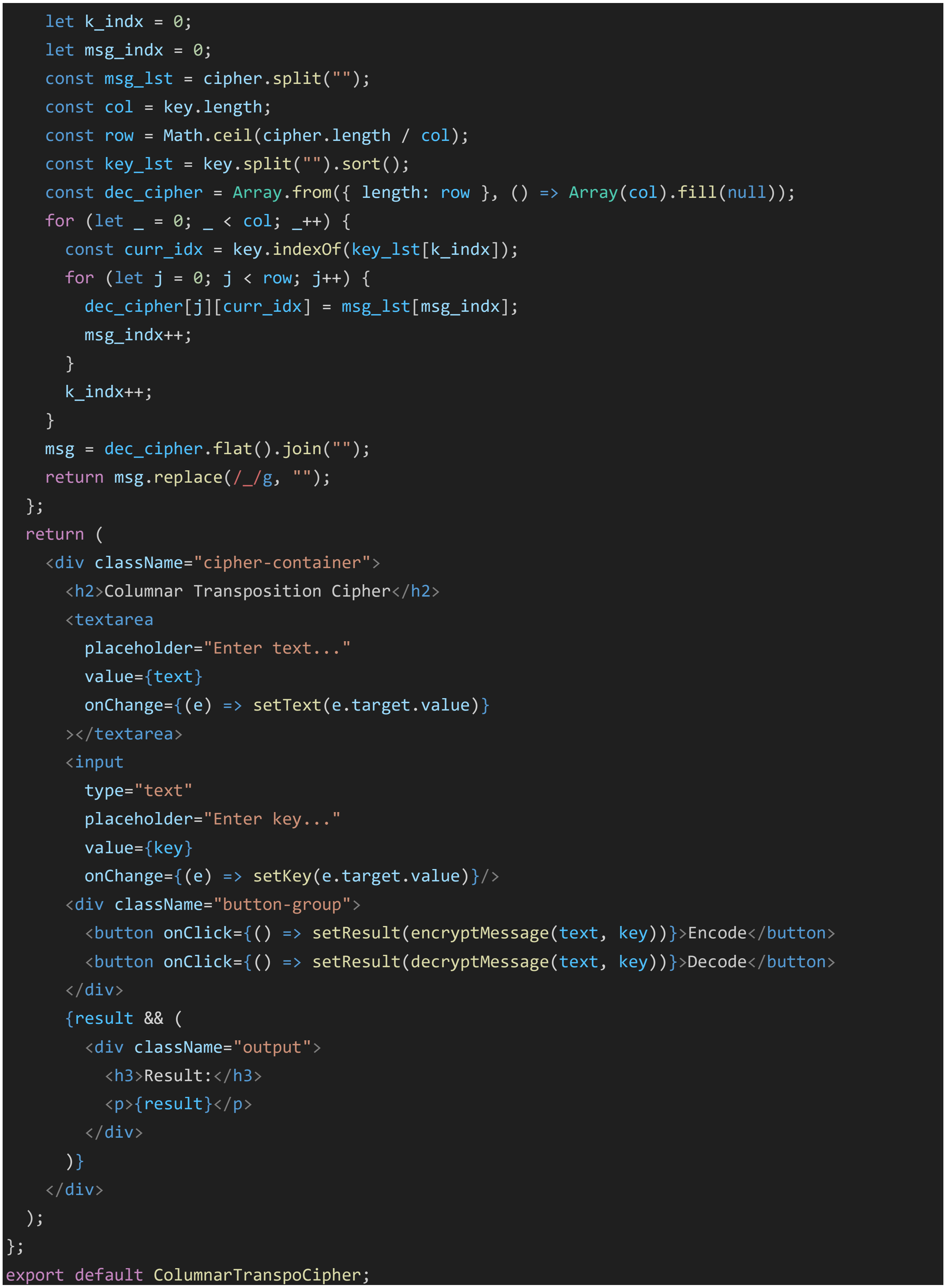
1. **Reconstruct the Grid:** Based on the known key, determine the number of columns.
2. **Fill in the Grid Column-wise:** Place the ciphertext back into columns according to the sorted key.
3. **Restore the Original Order:** Rearrange the columns to match the original key order.
4. **Read Row-wise:** Extract the plaintext message from the grid.

Implementation in React

A simple React-based implementation of the Columnar Transposition Cipher includes both **encryption and decryption** functionality:

Code:

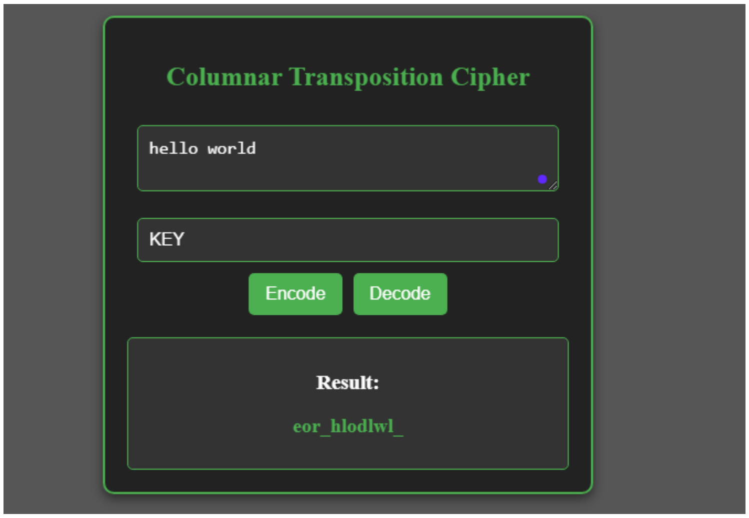
```
import React, { useState } from "react";
const ColumnarTranspoCipher = () => {
  const [text, setText] = useState("");
  const [key, setKey] = useState("");
  const [result, setResult] = useState("");
  const encryptMessage = (msg, key) => {
    let cipher = "";
    let k_indx = 0;
    const msg_lst = msg.replace(/\s/g, "").split("");
    const key_lst = key.split("").sort();
    const col = key.length;
    const row = Math.ceil(msg_lst.length / col);
    const fill_null = row * col - msg_lst.length;
    for (let i = 0; i < fill_null; i++) {
      msg_lst.push("_");
    }
    const matrix = [];
    for (let i = 0; i < msg_lst.length; i += col) {
      matrix.push(msg_lst.slice(i, i + col));
    }
    for (let _ = 0; _ < col; _++) {
      const curr_idx = key.indexOf(key_lst[k_indx]);
      for (const row of matrix) {
        cipher += row[curr_idx];
      }
      k_indx++;
    }
    return cipher;
  };
  const decryptMessage = (cipher, key) => {
    let msg = "";
```



```
    let k_indx = 0;
    let msg_indx = 0;
    const msg_lst = cipher.split("");
    const col = key.length;
    const row = Math.ceil(cipher.length / col);
    const key_lst = key.split("").sort();
    const dec_cipher = Array.from({ length: row }, () => Array(col).fill(null));
    for (let _ = 0; _ < col; _++) {
      const curr_idx = key.indexOf(key_lst[k_indx]);
      for (let j = 0; j < row; j++) {
        dec_cipher[j][curr_idx] = msg_lst[msg_indx];
        msg_indx++;
      }
      k_indx++;
    }
    msg = dec_cipher.flat().join("");
    return msg.replace(/_/g, "");
  };
  return (
    <div className="cipher-container">
      <h2>Columnar Transposition Cipher</h2>
      <textarea
        placeholder="Enter text..."
        value={text}
        onChange={(e) => setText(e.target.value)}
      ></textarea>
      <input
        type="text"
        placeholder="Enter key..."
        value={key}
        onChange={(e) => setKey(e.target.value)}/>
      <div className="button-group">
        <button onClick={() => setResult(encryptMessage(text, key))}>Encode</button>
        <button onClick={() => setResult(decryptMessage(text, key))}>Decode</button>
      </div>
      {result && (
        <div className="output">
          <h3>Result:</h3>
          <p>{result}</p>
        </div>
      )}
    </div>
  );
};
export default ColumnarTranspoCipher;
```

Output:

Encode:



Columnar Transposition Cipher

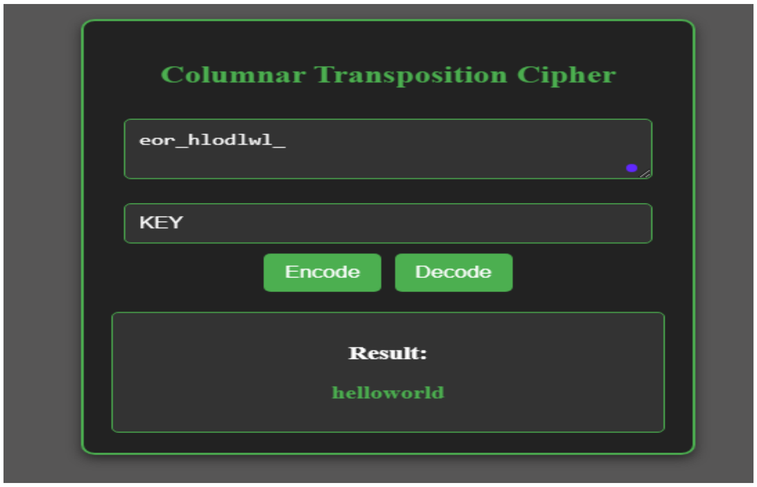
hello world

KEY

Result:

eor_hlodlwl_

Decode:



Columnar Transposition Cipher

eor_hlodlwl_

KEY

Result:

helloworld

Security Analysis

The Columnar Transposition Cipher offers **basic security**, but it is vulnerable to:

- **Frequency Analysis Attacks** (patterns in text remain intact)
- **Known-plaintext Attacks** (if part of the plaintext is known, key order can be guessed)
- **Brute Force Attacks** (short keys make it easier to crack)

Conclusion:

The Columnar Transposition Cipher is an excellent example of classical encryption techniques. While it is not secure by modern cryptographic standards, it provides an easy-to-understand approach to transposition-based encryption and decryption.