UNIVERSITY OF KARACHI

DEPARTMENT OF COMPUTER SCIENCE

UBIT

LAB MANUAL

Name: Rimsha Laraib

Seat No.: B21110006107

BSCS - IV

Section - A

Course Name: Network Security and Cryptography

Course Code: 631

Instructor: Abdul Bari

LAB # 01 : CAESAR CIPHER ALGORITHM

Object:

The Caesar cipher is a simple and well-known encryption technique used in cryptography. It is a type of substitution cipher where each letter in the plain text is shifted a fixed number of places down (or up) the alphabet..

Algorithm:

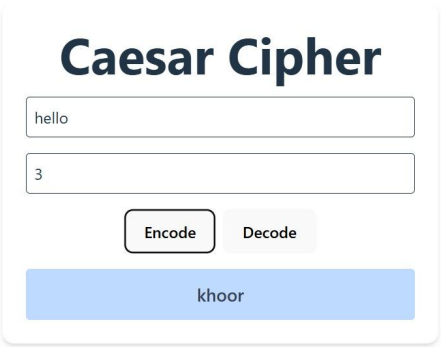
1. Choose a shift value (e.g., 3).
2. Replace each letter in the plain text with a letter that is shifted forward in the alphabet by the shift value.
3. If shifting past 'Z', wrap around to the beginning of the alphabet.

Code:

```
const caesarCipher = (str, shift, isEncode) => {

  return str

    .split("")

    .map((char) => {

      if (char.match(/[a-zA-Z]/)) {

        const base = char === char.toUpperCase() ? 65 : 97;

        const newShift = isEncode ? shift : -shift;

        return String.fromCharCode(

          ((char.charCodeAt(0) - base + newShift + 26) % 26) + base

        );}

      return char;

    })

    .join("");

};
```

Output:

Encode



Decode

Lab # 2 : One Time Pad Cipher

Object:

The One Time Pad (OTP) Cipher is a cryptographic technique that encrypts plaintext using a key of the same length. The key is used only once, making the cipher theoretically unbreakable if the key is truly random and kept secret.

Algorithm:

Encryption steps:

1. Input plain text message (Only uppercase letters) and key (Same length as plain text).
2. Convert each letter of the plain text and key into its corresponding numeric position (A=0, B=1, ..., Z=25).
3. Add the numeric values of the plain text and key.
4. If the result is greater than 25, subtract 26 to keep it within the range of alphabets.
5. Convert the result back into its corresponding letter.
6. Append the result to the encrypted text.

Decryption steps:

1. Input cipher text message (Only uppercase letters) and key.
2. Convert each letter of the cipher text and key into numeric values.
3. Subtract the key value from the cipher text value.
4. If the result is negative, add 26 to wrap around the alphabet.
5. Convert the result back into its corresponding letter.
6. Append the result to the plain text.

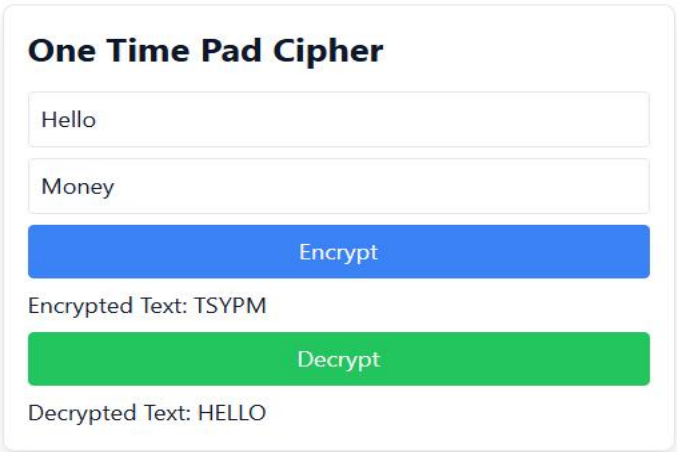
Code:

```
const stringEncryption = (text, key) => {
  let cipherText = "";
  let cipher = [];
  for (let i = 0; i < key.length; i++) {
    cipher[i] = text.charCodeAt(i) - 'A'.charCodeAt(0) + key.charCodeAt(i) - 'A'.charCodeAt(0);
  }
for (let i = 0; i < key.length; i++) {
    if (cipher[i] > 25) {
      cipher[i] = cipher[i] - 26;
```

```
    }}
for (let i = 0; i < key.length; i++) {
    let x = cipher[i] + 'A'.charCodeAt(0);
    cipherText += String.fromCharCode(x);
  } return cipherText; };
```

```
const stringDecryption = (s, key) => {
  let plainText = "";
  let plain = [];
  for (let i = 0; i < key.length; i++) {
    plain[i] = s.charCodeAt(i) - 'A'.charCodeAt(0) - (key.charCodeAt(i) - 'A'.charCodeAt(0));
  }
for (let i = 0; i < key.length; i++) {
    if (plain[i] < 0) {
      plain[i] = plain[i] + 26;
    }}
for (let i = 0; i < key.length; i++) {
    let x = plain[i] + 'A'.charCodeAt(0);
    plainText += String.fromCharCode(x);
  }return plainText;};
```

Output:



Lab # 3 : Rail Fence Cipher

Object:

The Rail Fence Cipher is a transposition cipher that arranges the plaintext in a zigzag pattern across multiple rails and then reads off the cipher text row by row.

Algorithm:

Encryption Steps:

1. Create a 2D array (matrix) with key rows and text.length columns.
2. Place characters in a zigzag pattern:
    1. Move down the rows until the last rail.
    2. Change direction and move up the rows until the first rail.
3. Read the matrix row by row to obtain the cipher text.

Decryption Steps:

1. Fill the matrix with placeholders (*) in the zigzag pattern.
2. Replace placeholders with the cipher text characters.
3. Traverse the matrix along the zigzag pattern to reconstruct the plaintext.

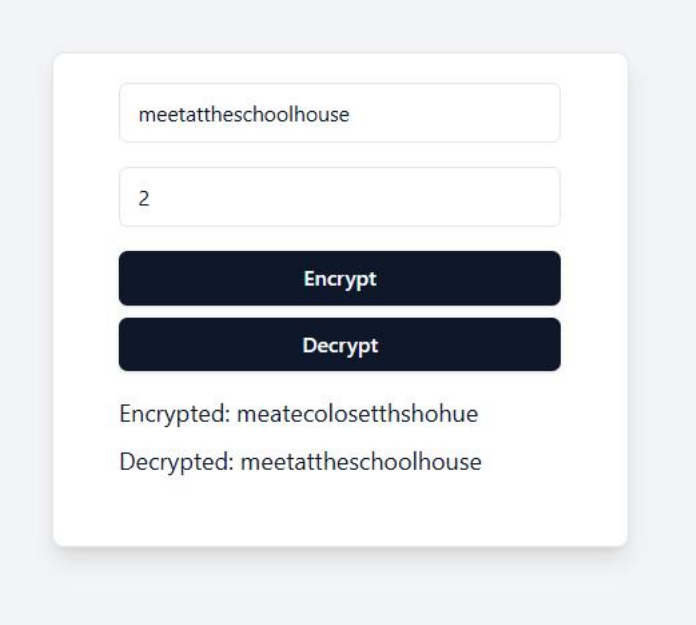
Code:

```
const encryptRailFence = (text, key) => {

  let rail = new Array(key).fill().map(() => new Array(text.length).fill('\n'));

  let dir_down = false;

  let row = 0, col = 0;



  for (let i = 0; i < text.length; i++) {

    if (row === 0 || row === key - 1) dir_down = !dir_down;

    rail[row][col++] = text[i];

    dir_down ? row++ : row--;

  }

  let result = '';

  for (let i = 0; i < key; i++) {

    for (let j = 0; j < text.length; j++) {
```

```
      if (rail[i][j] !== '\n') result += rail[i][j];
    }
  }
  return result;
};
```

```
const decryptRailFence = (cipher, key) => {
  let rail = new Array(key).fill().map(() => new Array(cipher.length).fill('\n'));
  let dir_down = false;
  let row = 0, col = 0;
  for (let i = 0; i < cipher.length; i++) {
    if (row === 0) dir_down = true;
    if (row === key - 1) dir_down = false;
    rail[row][col++] = '*';
    dir_down ? row++ : row--;
  }
  let index = 0;
  for (let i = 0; i < key; i++) {
    for (let j = 0; j < cipher.length; j++) {
      if (rail[i][j] === '*' && index < cipher.length) {
        rail[i][j] = cipher[index++];
      }}}
  let result = '';
  row = 0, col = 0;
  dir_down = false;
  for (let i = 0; i < cipher.length; i++) {
    if (row === 0) dir_down = true;
    if (row === key - 1) dir_down = false;
    if (rail[row][col] !== '*') result += rail[row][col++];
    dir_down ? row++ : row--;
  }
  return result;
};
```

Output:


meetattheschoolhouse
2
Encrypt
Decrypt
Encrypted: meatecolosetthshohue
Decrypted: meetattheschoolhouse

Lab # 4

Columnar Transposition Cipher

Object:

The columnar transposition cipher is a classical encryption method that rearranges the characters of the plaintext based on a keyword. It encrypts messages by writing them into a grid row by row and then reading the columns according to the alphabetical order of the keyword.

Algorithm:

Encryption Steps:

1. Take input a plaintext message and a keyword.
2. Write the message into a matrix row-wise, with the number of columns equal to the length of the key.
3. If the message length is not divisible by the key length, fill the remaining cells with a padding character (_).
4. Arrange the key alphabetically to determine the column order.
5. Read the matrix column-wise according to the sorted key order.
6. Concatenate the characters column by column to form the ciphertext.

Decryption steps:

1. Input ciphertext and the keyword.
2. Determine the number of rows based on the message length and key length.
3. Initialize an empty matrix.
4. Alphabetize the key to get the correct column order.
5. Place the ciphertext characters column-wise into the matrix according to the sorted key order.
6. Read the matrix row-wise to reconstruct the plaintext.
7. Strip any padding characters from the end of the message.

Code:

```
const encryptMessage = (msg, key) => {

  let cipher = "";

  let k_indx = 0;

  const msg_len = msg.length;

  const msg_lst = Array.from(msg);
```

```
  const key_lst = Array.from(key).sort();

  const col = key.length;

  const row = Math.ceil(msg_len / col);

  const fill_null = (row * col) - msg_len;



  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_len = cipher.length;

  const msg_lst = Array.from(cipher);

  const col = key.length;

  const row = Math.ceil(msg_len / col);

  const key_lst = Array.from(key).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('');
  const null_count = (msg.match(/_/g) || []).length;
 if (null_count > 0) {
    return msg.slice(0, -null_count);
  }
  return msg;
};
```

Output: