

**Name: Arham Sharif**

**Seat No.: EB21102022**

**Section: B**

**Subject: Network Security & Cryptography**

**Language: JavaScript**

# LAB#1

## CEASER CIPHER

### Introduction:

This program implements the Caesar cipher, a basic form of substitution cipher where each letter in the plaintext is shifted a certain number of places up or down in the alphabet. It provides a simple method of encrypting messages and can be easily decrypted if the shift value is known.

### Method of Encryption:

The Caesar cipher encrypts plaintext by shifting each letter in the message by a fixed number of positions to the right in the alphabet. For example, with a shift of 3, 'A' becomes 'D', 'B' becomes 'E', and so on. Both uppercase and lowercase letters are shifted, while non-alphabetic characters remain unchanged.

### Method of Decryption:

Decryption in the Caesar cipher involves shifting each letter in the encrypted message by the same number of positions to the left in the alphabet to retrieve the original plaintext. For example, with a shift of 3, 'D' becomes 'A', 'E' becomes 'B', and so on. Non-alphabetic characters remain unchanged during decryption.

### CODE:

```
const simpleInc = 3;

const simpleCharArr = Array.from({ length: 26 }, (_, i) =>
String.fromCharCode(97 + i));

const simpleLenCharArr = simpleCharArr.length;

""-----ENCODE-----""

# Function Can Encode Char

const encodeCharSimple = (char) => {
  let encodeChar = '';
  let index = -1;
  for (let i = 0; i < simpleLenCharArr; i++) {
    if (simpleCharArr[i] === char) {
```

```
        index = i + simpleInc;
        if (index >= simpleLenCharArr) {
            index %= simpleLenCharArr;
        }
        encodeChar = simpleCharArr[index];
        break;
    }
}
if (index != -1) {
    return encodeChar;
} else {
    return char;
}
}
```

**Output:**

## Ciphers

Simple Shifting

Enter Text:

Arham Sharif

EncodeDecode

Output:

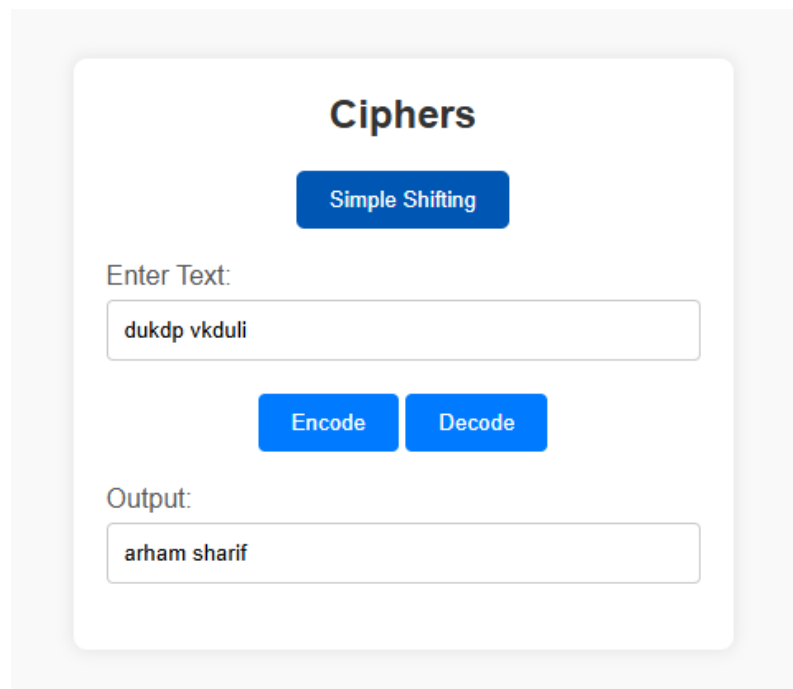
dukdp vkduli

### CODE:

```
""-----DECODE-----""
# Function to Decode Char
const decodeCharSimple = (char) => {
  let decodeChar = '';
  let index = -1;
  for (let i = 0; i < simpleLenCharArr; i++) {
    if (simpleCharArr[i] === char) {
      index = i - simpleInc;
      if (index < 0) {
        index += simpleLenCharArr;
      }
      decodeChar = simpleCharArr[index];
      break;
    }
  }
}
```

```
if (index !== -1) {  
    return decodeChar;  
} else {  
    return char;  
}  
}
```

### Output:



The screenshot shows a web application titled "Ciphers". Below the title is a blue button labeled "Simple Shifting". Underneath is a text input field labeled "Enter Text:" containing the text "dukdp vkduLi". Below the input field are two blue buttons: "Encode" and "Decode". At the bottom, there is an "Output:" label followed by a text output field containing the text "arham sharif".

## LAB#2

## OTP CIPHER

### Introduction:

This program implements the **One-Time Pad (OTP) cipher**, a theoretically unbreakable encryption method that uses a random key that is as long as the plaintext. Each letter in the plaintext is shifted by a

completely random amount determined by the corresponding character in the key. It offers perfect security when the key is truly random, used only once, and kept completely secret.

### **Method of Encryption:**

The OTP cipher encrypts plaintext by shifting each letter based on a completely random key of the same length. Each character in the plaintext is shifted forward by an amount determined by the corresponding character in the key. For example, if the key character is 'C' (position 2 in the alphabet), the plaintext letter is shifted by 2 positions. Both uppercase and lowercase letters are shifted, while non-alphabetic characters remain unchanged. The randomness and uniqueness of the key ensure maximum security.

### **Method of Decryption:**

Decryption in the OTP cipher involves reversing the encryption process using the same random key. Each letter in the cipher text is shifted backwards by the value of the corresponding letter in the key to retrieve the original plaintext. Since the key is truly random and used only once, the decryption process perfectly reconstructs the original message. Non-alphabetic characters remain unchanged during decryption.

#### **CODE:**

```
const otpCharArr = Array.from({ length: 26 }, (_, i) =>
String.fromCharCode(97 + i));

// Function to generate a random OTP key of the same length as the
message
const generateOtpKey = (length) => {
  const charset = otpCharArr.join("");
  let key = '';
  for (let i = 0; i < length; i++) {
    const randomIndex = Math.floor(Math.random() * charset.length);
    key += charset[randomIndex];
  }
  return key;
};

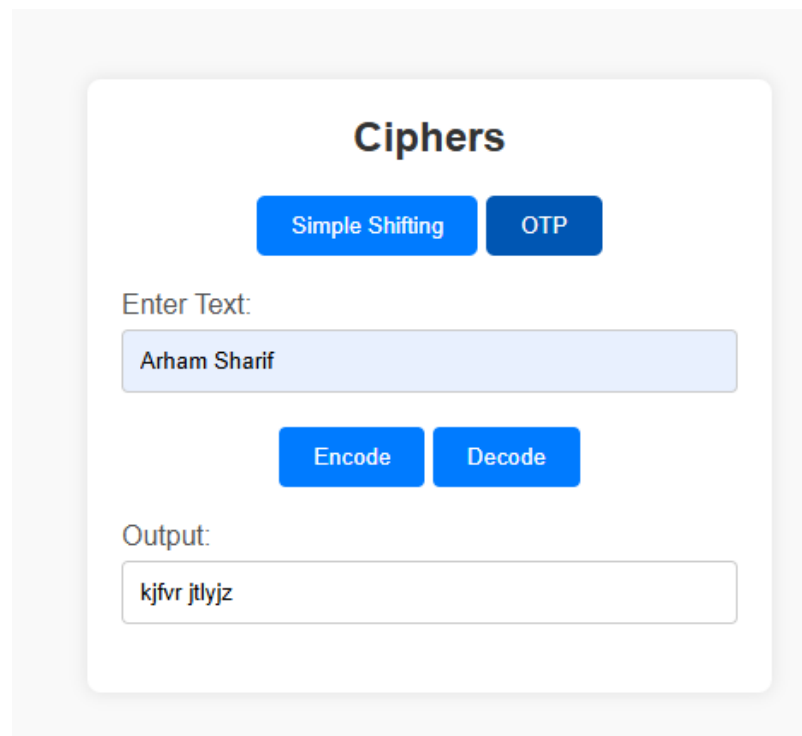
// Function to save OTP key to local storage
const saveOtpKeyToLocalStorage = (key) => {
  localStorage.setItem('otpKey', key);
};

// Function to retrieve OTP key from local storage
```

```
const getOtpKeyFromLocalStorage = () => {
  return localStorage.getItem('otpKey');
};

// Function to encrypt message using OTP
const encryptOtp = (message, key) => {
  let result = '';
  for (let i = 0; i < message.length; i++) {
    const char = message.charAt(i);
    if (otpCharArr.includes(char)) {
      const messageIndex = otpCharArr.indexOf(char);
      const keyIndex = otpCharArr.indexOf(key.charAt(i));
      const encryptedChar = otpCharArr[(messageIndex + keyIndex) %
otpCharArr.length];
      result += encryptedChar;
    } else {
      result += char; // Non-alphabet characters remain unchanged
    }
  }
  return result;
};
```

### **Output:**

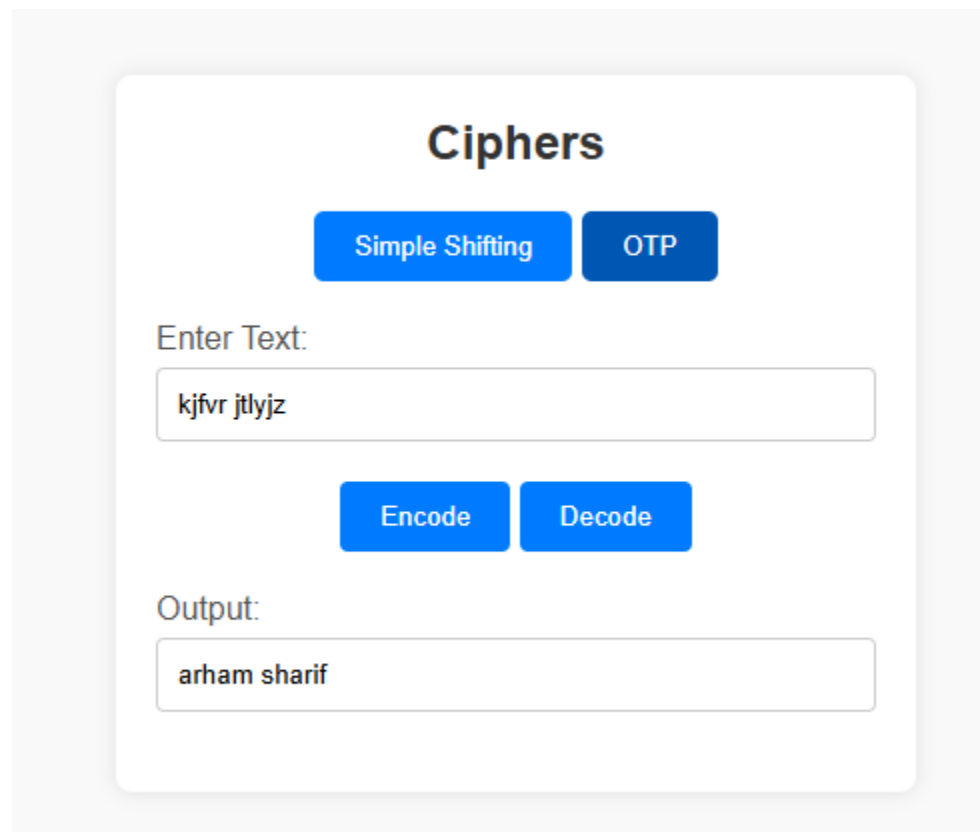


The screenshot shows a web application titled "Ciphers". It has two buttons at the top: "Simple Shifting" and "OTP". The "OTP" button is selected. Below these buttons is a label "Enter Text:" followed by a text input field containing "Arham Sharif". Below the input field are two buttons: "Encode" and "Decode". The "Encode" button is selected. Below these buttons is a label "Output:" followed by a text input field containing "kjfv r jtlyjz".

### CODE:

```
// Function to decrypt message using OTP
const decryptOtp = (message, key) => {
  let result = '';
  for (let i = 0; i < message.length; i++) {
    const char = message.charAt(i);
    if (otpCharArr.includes(char)) {
      const messageIndex = otpCharArr.indexOf(char);
      const keyIndex = otpCharArr.indexOf(key.charAt(i));
      const decryptedChar = otpCharArr[(messageIndex - keyIndex +
otpCharArr.length) % otpCharArr.length];
      result += decryptedChar;
    } else {
      result += char; // Non-alphabet characters remain unchanged
    }
  }
  return result;
};
```

### Output:



The screenshot shows a web application titled "Ciphers". It has two tabs: "Simple Shifting" and "OTP", with "OTP" being the active tab. Below the tabs, there is a label "Enter Text:" followed by a text input field containing the ciphertext "kjivr jtlyjz". Underneath the input field are two buttons: "Encode" and "Decode". The "Decode" button is highlighted, indicating it was clicked. Below the buttons, there is a label "Output:" followed by a text output field displaying the plaintext "arham sharif".



# LAB#3

## RAIL FENCE CIPHER

### Introduction:

This program implements the Rail Fence cipher, a transposition cipher that rearranges the characters of the plaintext into a zigzag pattern across a number of "rails". It offers basic security by altering the order of characters in the message.

### Method of Encryption:

The Rail Fence cipher encrypts plaintext by writing it in a zigzag pattern across a specified number of rails. Each character of the plaintext is written into successive rails, moving up and down, until the entire message is encoded. The cipher text is then read row by row to produce the encrypted message.

### Method of Decryption:

Decryption in the Rail Fence cipher involves reconstructing the zigzag pattern used during encryption. The cipher text is written into the corresponding rails based on the same zigzag pattern, allowing the original plaintext to be retrieved by reading the characters in the order they were originally written.

### CODE:

```
// Encrypt using Rail Fence Cipher
function encryptRailFence(text, rails) {
  if (rails <= 1) return text;

  const fence = Array.from({ length: rails }, () => []);
  let rail = 0;
  let direction = 1; // 1 = down, -1 = up

  for (const element of text) {
    fence[rail].push(element);
    rail += direction;

    if (rail === 0 || rail === rails - 1) {
      direction *= -1;
    }
  }
}
```

```
    return fence.flat().join('');  
}
```

### **Output:**

**Ciphers**

Simple Shifting   OTP   Rail Fence

Enter Text:

Arham Sharif

Enter Rails:

5

Encode   Decode

Output:

aarhrhsia fm

### **CODE:**

```
// Decrypt using Rail Fence Cipher  
function decryptRailFence(cipher, rails) {  
    if (rails <= 1) return cipher;  
  
    // Step 1: Create an empty matrix with placeholders  
    const pattern = Array.from({ length: rails }, () =>  
        Array(cipher.length).fill(null));  
    let rail = 0;  
    let direction = 1;  
  
    for (let col = 0; col < cipher.length; col++) {
```

```

    pattern[rail][col] = '*';
    rail += direction;

    if (rail === 0 || rail === rails - 1) {
        direction *= -1;
    }
}

// Step 2: Fill the pattern with actual characters
let index = 0;
for (let r = 0; r < rails; r++) {
    for (let c = 0; c < cipher.length; c++) {
        if (pattern[r][c] === '*' && index < cipher.length) {
            pattern[r][c] = cipher[index++];
        }
    }
}

// Step 3: Read the message by zigzag
let result = '';
rail = 0;
direction = 1;

for (let col = 0; col < cipher.length; col++) {
    result += pattern[rail][col];
    rail += direction;

    if (rail === 0 || rail === rails - 1) {
        direction *= -1;
    }
}

return result;
}

```

### **Output:**

## Ciphers

Simple Shifting

OTP

Rail Fence

Enter Text:

aarhrhsia fm

Enter Rails:

5

Encode

Decode

Output:

arham sharif

## LAB#4

### PLAYFAIR CIPHER

#### Introduction:

This program implements the Playfair cipher, a digraph substitution cipher that operates on pairs of characters. It uses a 5x5 grid of letters derived from a keyword to encrypt and decrypt messages.

#### Method of Encryption:

The Playfair cipher encrypts plaintext by first processing it into digraphs (pairs of characters). Each digraph is then mapped to a corresponding pair of cipher text characters based on their positions in the Playfair grid. If the characters of a digraph are in the same row, they are replaced with the characters immediately to their right (wrapping around to the beginning if necessary). If they are in the same

column, they are replaced with the characters directly below them. If they form a rectangle, they are replaced with the characters on the same row, but at the opposite corners of the rectangle.

### **Method of Decryption:**

Decryption in the Playfair cipher involves reversing the encryption process. Each digraph in the cipher text is mapped back to its corresponding plaintext digraph using the positions of the characters in the Playfair grid. Each pair of cipher text characters is decrypted based on whether they are in the same row, column, or form a rectangle, thereby reconstructing the original plaintext.

#### **CODE:**

```
const alphabetArr = Array.from({ length: 26 }, (_, i) =>
String.fromCharCode(65 + i))
    .filter(c => c !== 'J') // remove 'J'
    .join('');

// Generate random Playfair key (5x5 grid)
function generatePlayfairKey() {
    const shuffled = [...alphabetArr];
    for (let i = shuffled.length - 1; i > 0; i--) {
        const j = Math.floor(Math.random() * (i + 1));
        [shuffled[i], shuffled[j]] = [shuffled[j], shuffled[i]];
    }
    return shuffled.join('');
}

// Save key to localStorage
function savePlayfairKey(key) {
    localStorage.setItem('playfairKey', key);
}

// Get or generate key from localStorage
function getPlayfairKey() {
    let key = localStorage.getItem('playfairKey');
    if (!key) {
        key = generatePlayfairKey();
        savePlayfairKey(key);
    }
    return key;
}

// Create 5x5 key matrix from key
```

```

function createMatrix(key) {
  const matrix = [];
  for (let i = 0; i < 25; i += 5) {
    matrix.push(key.slice(i, i + 5).split(''));
  }
  return matrix;
}

// Find letter position in key matrix
function findPosition(matrix, letter) {
  for (let row = 0; row < 5; row++) {
    const col = matrix[row].indexOf(letter);
    if (col !== -1) return { row, col };
  }
  return null;
}

// Prepare text for Playfair cipher (remove non-letters, replace J
with I, make pairs)
function prepareText(text) {
  text = text.toUpperCase().replace(/[^A-Z]/g, '').replace(/J/g,
'I');
  let result = '';
  for (let i = 0; i < text.length; i += 2) {
    let a = text[i];
    let b = text[i + 1] || 'X';
    if (a === b) {
      result += a + 'X';
      i--;
    } else {
      result += a + b;
    }
  }
  return result;
}

// Encrypt a pair of letters
function encryptPair(a, b, matrix) {
  const posA = findPosition(matrix, a);
  const posB = findPosition(matrix, b);
  if (posA.row === posB.row) {
    return matrix[posA.row][(posA.col + 1) % 5] +
matrix[posB.row][(posB.col + 1) % 5];

```

```

    } else if (posA.col === posB.col) {
      return matrix[(posA.row + 1) % 5][posA.col] + matrix[(posB.row +
1) % 5][posB.col];
    } else {
      return matrix[posA.row][posB.col] + matrix[posB.row][posA.col];
    }
  }
}

// Encrypt full message
function encryptPlayfair(message) {
  const key = getPlayfairKey();
  const matrix = createMatrix(key);
  const prepared = prepareText(message);
  let encrypted = '';
  for (let i = 0; i < prepared.length; i += 2) {
    encrypted += encryptPair(prepared[i], prepared[i + 1], matrix);
  }
  return encrypted;
}

```

### **Output:**

The screenshot shows a web application titled "Ciphers". It has four buttons: "Simple Shifting", "OTP", "Rail Fence", and "PlayFair". The "PlayFair" button is selected. Below the buttons is a text input field labeled "Enter Text:" containing the text "Arham Sharif". Below the input field are two buttons: "Encode" and "Decode". Below these buttons is another text input field labeled "Output:" containing the encrypted text "cnpchgipcvszp".

### **CODE:**

```

// Decrypt a pair of letters
function decryptPair(a, b, matrix) {

```

```

    const posA = findPosition(matrix, a);
    const posB = findPosition(matrix, b);
    if (posA.row === posB.row) {
        return matrix[posA.row][(posA.col + 4) % 5] +
matrix[posB.row][(posB.col + 4) % 5];
    } else if (posA.col === posB.col) {
        return matrix[(posA.row + 4) % 5][posA.col] + matrix[(posB.row +
4) % 5][posB.col];
    } else {
        return matrix[posA.row][posB.col] + matrix[posB.row][posA.col];
    }
}

// Decrypt full message
function decryptPlayfair(cipherText) {
    const key = getPlayfairKey();
    const matrix = createMatrix(key);
    let decrypted = '';
    for (let i = 0; i < cipherText.length; i += 2) {
        decrypted += decryptPair(cipherText[i], cipherText[i + 1],
matrix);
    }
    return decrypted;
}

```

## **Output:**

**Ciphers**

Simple Shifting

OTP

Rail Fence

PlayFair

Enter Text:

cnp c g i p c v s z p

Encode

Decode

Output:

arhamsharifx



# LAB#5

## VIGINERERE CIPHER

### Introduction:

This program implements the Vigenère cipher, a polyalphabetic substitution cipher that uses a keyword to shift letters in the plaintext by varying amounts across different positions. It offers improved security compared to the Caesar cipher by using a keyword to determine multiple shift values.

### Method of Encryption:

The Vigenère cipher encrypts plaintext by shifting each letter in the message based on a keyword. The keyword determines the amount of shift applied to each letter in the plaintext cyclically. For example, if the keyword is 'KEY' and the plaintext is 'HELLO', the first letter 'H' is shifted by 'K', 'E' by 'E', 'L' by 'Y', and so on. Both uppercase and lowercase letters are shifted, while non-alphabetic characters remain unchanged.

### Method of Decryption:

Decryption in the Vigenère cipher involves reversing the encryption process using the same keyword. Each letter in the cipher text is shifted backwards by the corresponding letter in the keyword to retrieve the original plaintext. Non-alphabetic characters remain unchanged during decryption.

### CODE:

```
// Function to generate a random Vigenère key of given length
const vigenereCharArr = Array.from({ length: 26 }, (_, i) =>
String.fromCharCode(97 + i));
function generateVigenereRandomKey(length) {
  const charset = vigenereCharArr.join("");
  let key = '';
  for (let i = 0; i < length; i++) {
    const randomIndex = Math.floor(Math.random() * charset.length);
    key += charset[randomIndex];
  }
  return key;
}
```

```
// Function to generate the Vigenère character array based on the key
function generateVigenereCharArr(key) {
  const charArr = [];
  for (let i = 0; i < key.length; i++) {
    const shift = key.charCodeAt(i) - 97; // Get the shift amount for
    each character in the key
    const shiftedChars =
    vigenereCharArr.slice(shift).concat(vigenereCharArr.slice(0, shift));
    charArr.push(shiftedChars);
  }
  return charArr;
}
```

```
// Function to save Vigenère key and character array to local storage
function saveVigenereToLocalStorage(key, charArr) {
  localStorage.setItem('vigenereKey', key);
  localStorage.setItem('vigenereCharArr', JSON.stringify(charArr));
}
```

```
// Function to retrieve Vigenère key and character array from local
storage
function getVigenereFromLocalStorage() {
  const key = localStorage.getItem('vigenereKey');
  const charArr =
  JSON.parse(localStorage.getItem('vigenereCharArr'));
  return { key, charArr };
}
```

```
// Generate random key and character array
const randomKey = generateVigenereRandomKey(6); // Change the length
as needed
const randomCharArr = generateVigenereCharArr(randomKey);
```

```
// Save them to local storage
saveVigenereToLocalStorage(randomKey, randomCharArr);
```

```
// Function to encrypt message using Vigenère shifting
function encryptVigenereShifting(message) {
  const { key, charArr } = getVigenereFromLocalStorage();

  let result = '';

  for (let i = 0, j = 0; i < message.length; i++) {
```

```

const c = message.charAt(i);
const index = vigenereCharArr.indexOf(c);
if (index !== -1) {
    result += charArr[j % key.length][index];
    j++;
} else {
    result += c;
}
}
return result;
}

```

### **Output:**

**Ciphers**

Simple Shifting

OTP

Rail Fence

PlayFair

Vigenère

Enter Text:

Arham Sharif

Encode

Decode

Output:

ntvkr fucfsk

### **CODE:**

```

// Function to decrypt message using Vigenère shifting
function decryptVigenereShifting(message) {
    const { key, charArr } = getVigenereFromLocalStorage();

    let result = '';

    for (let i = 0, j = 0; i < message.length; i++) {
        const c = message.charAt(i);
        const rowIndex = j % key.length;
        const charIndex = charArr[rowIndex].indexOf(c);

```

```
    if (charIndex !== -1) {  
        result += vigenereCharArr[charIndex];  
        j++;  
    } else {  
        result += c;  
    }  
}  
return result;  
}
```

**Output:**

## Ciphers

Simple Shifting

OTP

Rail Fence

PlayFair

Vigenère

Enter Text:

ntvkr fucfsk

Encode

Decode

Output:

arham sharif