Учреждение образования

«Белорусский государственный технологический университет»

**Кафедра информационных систем и технологий**

**«Отчёт по лабораторной работе №5»**

«Исследование криптографических шифров на основе перестановки символов»

**Выполнил:** студент 3 курса

4 группы специальности ПОИТ

Супрунюк Евгений Андреевич

**Проверил:** преподаватель

Сазонова Дарья Владимировна

Минск 2023

**Выполнять зашифрование и расшифрование маршрутной перестановкой (маршрут – зигзагом):**

Реализация на языке JavaScript:

|  |
| --- |
| function encryptByRoutePermutation(text) {      const textLength = text.length;      const columns = 3;      const rows = Math.ceil(textLength / columns);      const matrix = new Array(rows).fill().map(() => new Array(columns).fill(''));      let row = 0;      let col = 0;      let pos = 0      for (let col = 0; col < columns; col++) {          for (let row = 0; row < rows; row++) {              matrix[row][col] = text[pos++];              // if (row == 0) {              //     console.log();              // }              // process.stdout.write(matrix[row][col]);              // console.log('Row = ' + row + ', Column = ' + col + ', Symbol = ' + matrix[row][col]);          }      }      for (let col = 0; col < columns; col++) {          for (let row = 0; row < rows; row++) {              if (row == 0) {                  console.log();              }              process.stdout.write(matrix[row][col]);          }      }      console.log();      let result = '';      for (let row = 0; row < rows; row++) {          // console.log('Row = ' + row);          if (row % 2 == 1) {              for (let col = 0; col < columns; col++) {                  result += matrix[row][col];                  // console.log('Row = ' + row + ', Column = ' + col + ', Symbol = ' + matrix[row][col]);              }          }          else {              for (let col = columns - 1; col >= 0; col--) {                  result += matrix[row][col];                  // console.log('Row = ' + row + ', Column = ' + j + ', Symbol = ' + matrix[row][j]);              }          }      }      return result;  } |

|  |
| --- |
| function decryptByRoutePermutation(text) {      const textLength = text.length;      const columns = 3;      const rows = Math.ceil(textLength / columns);      const matrix = new Array(rows).fill().map(() => new Array(columns).fill(''));      let row = 0;      let col = 0;      let pos = 0      for (let row = 0; row < rows; row++) {          for (let col = 0; col < columns; col++) {              matrix[row][col] = text[pos++];          }      }      for (let col = 0; col < columns; col++) {          for (let row = 0; row < rows; row++) {              if (row == 0) {                  console.log();              }              process.stdout.write(matrix[row][col]);          }      }      console.log();      const newMatrix = new Array(rows).fill().map(() => new Array(columns).fill(''));        for (let row = rows - 1; row >= 0; row--) {          // console.log('Row = ' + row);          if (row % 2 == 1) {              for (let col = 0; col < columns; col++) {                  newMatrix[row][col] += matrix[row][col];              }          }          else {              for (let col = columns - 1, ncol = 0; col >= 0; col--, ncol++) {                  newMatrix[row][ncol] += matrix[row][col];              }          }      }      let result = '';      for (let col = 0; col < columns; col++) {          for (let row = 0; row < rows; row++) {              result += newMatrix[row][col];          }      }      return result;  } |

**Выполнять зашифрование/расшифрование множественной перестановкой, ключевые слова – собственные имя и фамилия:**

|  |
| --- |
| function encryptMultiplePermutation(plaintext, keyword1, keyword2) {      let ciphertext = '';      for (let chunk = 0; chunk < plaintext.length / (keyword1.length \* keyword2.length); chunk++) {          let matrix = [];          for (let i = 0; i < keyword2.length; i++) {              matrix.push([]);              for (let j = 0; j < keyword1.length; j++) {                  matrix[i][j] = i \* keyword1.length + j < Math.min((keyword1.length \* keyword2.length), plaintext.length - chunk \* keyword1.length \* keyword2.length) ?                      plaintext[chunk \* keyword1.length \* keyword2.length + i \* keyword1.length + j] : ' ';              }          }          for (let i = 0; i < matrix.length; i++) {              console.log(matrix[i]);          }          console.log();          matrix = [...keyword2].map((value, index) => ({ index: index, letter: value }))              .sort((a, b) => a.letter === b.letter ? 0 : (a.letter < b.letter ? -1 : 1))              .map((value, index) => ({ index: value.index, array: matrix[index] }))              .sort((a, b) => a.index === b.index ? 0 : (a.index < b.index ? -1 : 1))              .map(value => value.array);          for (let i = 0; i < matrix.length; i++) {              console.log(matrix[i]);          }          console.log();          matrix = [...keyword1].map((value, index) => ({ index: index, letter: value }))              .sort((a, b) => a.letter === b.letter ? 0 : (a.letter < b.letter ? -1 : 1))              .map((value, index) => ({ index: value.index, array: matrix.map(row => row[index]) }))              .sort((a, b) => a.index === b.index ? 0 : (a.index < b.index ? -1 : 1))              .map(value => value.array);          for (let i = 0; i < matrix.length; i++) {              console.log(matrix[i]);          }          console.log();          for (let i = 0; i < keyword2.length; i++) {              for (let j = 0; j < keyword1.length; j++) {                  ciphertext += matrix[i][j];              }          }      }      return ciphertext;  } |

|  |
| --- |
| function decryptMultiplePermutation(ciphertext, keyword1, keyword2) {      let plaintext = '';      for (let chunk = 0; chunk < ciphertext.length / (keyword1.length \* keyword2.length); chunk++) {          let matrix = [];          for (let i = 0; i < keyword2.length; i++) {              matrix.push([]);              for (let j = 0; j < keyword1.length; j++) {                  matrix[i][j] = ciphertext[chunk \* keyword1.length \* keyword2.length + i \* keyword1.length + j]              }          }          matrix = [...keyword1].map((value, index) => ({ letter: value, index: index }))              .map((value, index) => ({ letter: value.letter, array: matrix[index] }))              .sort((a, b) => a.letter === b.letter ? 0 : (a.letter < b.letter ? -1 : 1))              .map(value => value.array);          matrix = [...keyword2].map((value, index) => ({ letter: value, index: index }))              .map((value, index) => ({ letter: value.letter, array: matrix.map(row => row[index]) }))              .sort((a, b) => a.letter === b.letter ? 0 : (a.letter < b.letter ? -1 : 1))              .map(value => value.array);          for (let i = 0; i < keyword2.length; i++) {              for (let j = 0; j < keyword1.length; j++) {                  plaintext += matrix[i][j];              }          }      }      return plaintext;  } |