Учреждение образования

«Белорусский государственный технологический университет»

**Кафедра информационных систем и технологий**

**«Отчёт по лабораторной работе №3»**

“Исследование криптографических шифров на основе перестановки символов”

**Выполнил:** студент 4 курса

1 группы специальности ИСИт

Палазник Арсений Викторович

**Проверил:** преподаватель

Сазонова Дарья Владимировна

Минск 2024

**Разработать авторское приложение в соответствии с целью лабораторной работы. Приложение должно реализовывать следующие операции:**

* **выполнять зашифрование/расшифрование текстовых документов (объемом не менее 500 знаков), созданных на основе алфавита языка в соответствии с нижеследующей таблицей вариантов задания; при этом следует использовать шифры подстановки из третьего столбца данной таблицы (варианты задания в табл. 3.11);**

Реализация приложения:

|  |
| --- |
| // Функция для шифрования маршрутной перестановкой змейкой  function snakeRouteCipherEncrypt(text) {  const columns = 5; // Пример количества столбцов  const rows = Math.ceil(text.length / columns);  let table = [];  // Запись текста змейкой по строкам  for (let i = 0; i < rows; i++) {  let rowText = text.slice(i \* columns, (i + 1) \* columns);  if (i % 2 === 1) {  rowText = rowText.split("").reverse().join(""); // Переворачиваем строку для четных строк  }  table.push(rowText);  }  // Считывание по строкам  let encryptedText = "";  for (let i = 0; i < rows; i++) {  encryptedText += table[i];  }  return encryptedText;  }  // Функция для расшифровки маршрутной перестановки змейкой  function snakeRouteCipherDecrypt(encryptedText) {  const columns = 5;  const rows = Math.ceil(encryptedText.length / columns);  let table = [];  // Заполняем таблицу змейкой по строкам  for (let i = 0; i < rows; i++) {  let rowText = encryptedText.slice(i \* columns, (i + 1) \* columns);  if (i % 2 === 1) {  rowText = rowText.split("").reverse().join(""); // Переворачиваем строку для четных строк  }  table.push(rowText);  }  // Преобразуем таблицу в строку  let decryptedText = "";  for (let i = 0; i < rows; i++) {  decryptedText += table[i];  }  return decryptedText;  }  // Основная функция для шифрования  function encrypt() {  const method = document.getElementById("method").value;  const text = document.getElementById("inputText").value;  const key = document.getElementById("key").value;  let result = "";  if (method === "route") {  result = snakeRouteCipherEncrypt(text);  } else if (method === "multiple") {  result = multipleCipherEncrypt(text, "Влад", "Вакуленчик");  }  document.getElementById("outputText").value = result;  }  // Функция для расшифровки  function decrypt() {  const method = document.getElementById("method").value;  const text = document.getElementById("outputText").value;  let result = "";  if (method === "route") {  result = snakeRouteCipherDecrypt(text);  } else if (method === "multiple") {  result = multipleCipherDecrypt(text, "Влад", "Вакуленчик");  }  document.getElementById("outputText").value = result;  }  // Получить порядок перестановки на основе ключевого слова  function getPermutationOrder(key) {  const order = key.split("").map((char, index) => ({ char, index }));  order.sort((a, b) => a.char.localeCompare(b.char)); // Сортируем по алфавиту  return order.map((item) => item.index);  }  // Восстановление исходного порядка на основе ключевого слова  function getDecryptionOrder(key) {  const order = getPermutationOrder(key);  const decryptionOrder = [];  for (let i = 0; i < order.length; i++) {  decryptionOrder[order[i]] = i;  }  return decryptionOrder;  }  // Шифрование с использованием множественной перестановки  function multipleCipherEncrypt(text, rowKey, colKey) {  const rowLength = rowKey.length; // Количество строк  const colLength = colKey.length; // Количество столбцов  const totalLength = rowLength \* colLength;  // Добавляем пробелы, если текст короче, чем нужно для заполнения матрицы  if (text.length < totalLength) {  text = text.padEnd(totalLength, " ");  }  // Разделение текста на строки  let table = [];  for (let i = 0; i < rowLength; i++) {  table.push(text.slice(i \* colLength, (i + 1) \* colLength).split(""));  }  // Перестановка строк по ключу rowKey  const rowOrder = getPermutationOrder(rowKey);  let permutedRows = rowOrder.map((index) => table[index]);  // Перестановка столбцов по ключу colKey  const colOrder = getPermutationOrder(colKey);  let encryptedTable = permutedRows.map((row) =>  colOrder.map((index) => row[index])  );  // Преобразуем обратно в строку  let encryptedText = encryptedTable.flat().join("");  return encryptedText;  }  // Расшифровка с использованием множественной перестановки  function multipleCipherDecrypt(encryptedText, rowKey, colKey) {  const rowLength = rowKey.length; // Количество строк  const colLength = colKey.length; // Количество столбцов  // Разделение зашифрованного текста на строки  let table = [];  for (let i = 0; i < rowLength; i++) {  table.push(  encryptedText.slice(i \* colLength, (i + 1) \* colLength).split("")  );  }  // Восстановление исходного порядка столбцов по ключу colKey  const colOrder = getDecryptionOrder(colKey);  let permutedCols = table.map((row) => colOrder.map((index) => row[index]));  // Восстановление исходного порядка строк по ключу rowKey  const rowOrder = getDecryptionOrder(rowKey);  let decryptedTable = rowOrder.map((index) => permutedCols[index]);  // Преобразуем обратно в строку  let decryptedText = decryptedTable.flat().join("");  return decryptedText;  } |

Результат выполнения:



