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

Мозолевский Александр Дмитриевич

**Проверил:** преподаватель

Сазонова Дарья Владимировна

Минск 2023

**Маршрутная перестановка (маршрут – змейкой; параметры таблицы – по указанию преподавателя)**

|  |
| --- |
| package route  import "strings"  func Encrypt(input string, rows int, cols int) string {  grid := make([][]rune, rows)  for i := range grid {  grid[i] = make([]rune, cols)  }  x, y := 0, 0  dir := 1  for \_, char := range input {  grid[x][y] = char  y += dir  if y < 0 || y >= cols {  dir = -dir  y += dir  x++  }  }  var output strings.Builder  for i := 0; i < rows; i++ {  for j := 0; j < cols; j++ {  if grid[i][j] != 0 {  output.WriteRune(grid[i][j])  }  }  }  return output.String()  }  func Decrypt(input string, rows int, cols int) string {  grid := make([][]rune, rows)  for i := range grid {  grid[i] = make([]rune, cols)  }  for i := range grid {  for j := range grid[i] {  grid[i][j] = ' '  }  }  pos := 0  for i := 0; i < rows; i++ {  for j := 0; j < cols; j++ {  if pos < len(input) {  grid[i][j] = rune(input[pos])  pos++  }  }  }  var output strings.Builder  x, y := 0, 0  dir := 1  for i := 0; i < rows\*cols; i++ {  output.WriteRune(grid[x][y])  y += dir  if y < 0 || y >= cols {  dir = -dir  y += dir  x++  }  }  return output.String()  } |

**Множественная перестановка, ключевые слова – собственные имя и фамилия**

|  |
| --- |
| package mpr  import (  "strings"  )  // Encrypt encrypts plaintext using the multiple transposition algorithm with the given key  func Encrypt(plaintext string, key string) string {  // Remove any spaces from the key  key = strings.ReplaceAll(key, " ", "")  // Calculate the number of rows and columns needed for the transposition grid  rows := (len(plaintext) + len(key) - 1) / len(key)  cols := len(key)  // Add padding to the plaintext if necessary  plaintext += strings.Repeat("X", rows\*cols-len(plaintext))  // Create the transposition grid and fill it with the padded plaintext  grid := make([][]rune, rows)  for i := range grid {  grid[i] = make([]rune, cols)  for j := range grid[i] {  grid[i][j] = rune(plaintext[i\*cols+j])  }  }  // Create a map to keep track of the original column order  colOrder := make(map[int]int)  for i, c := range key {  colOrder[i] = strings.IndexRune(key, c)  }  // Sort the column order by its values  sortedOrder := make([]int, cols)  for i := range sortedOrder {  sortedOrder[i] = colOrder[i]  }  for i := 0; i < cols; i++ {  for j := i + 1; j < cols; j++ {  if sortedOrder[i] > sortedOrder[j] {  sortedOrder[i], sortedOrder[j] = sortedOrder[j], sortedOrder[i]  }  }  }  // Create a new transposition grid with the sorted column order  sortedGrid := make([][]rune, rows)  for i := range sortedGrid {  sortedGrid[i] = make([]rune, cols)  for j := range sortedGrid[i] {  sortedGrid[i][j] = grid[i][colOrder[sortedOrder[j]]]  }  }  // Read out the encrypted message from the sorted transposition grid column by column  ciphertext := ""  for j := 0; j < cols; j++ {  for i := 0; i < rows; i++ {  ciphertext += string(sortedGrid[i][j])  }  }  return ciphertext  }  // Decrypt decrypts ciphertext using the multiple transposition algorithm with the given key  func Decrypt(ciphertext string, key string) string {  // Remove any spaces from the key  key = strings.ReplaceAll(key, " ", "")  // Calculate the number of rows and columns needed for the transposition grid  rows := (len(ciphertext) + len(key) - 1) / len(key)  cols := len(key)  // Create the transposition grid and fill it with the ciphertext  grid := make([][]rune, rows)  for i := range grid {  grid[i] = make([]rune, cols)  for j := range grid[i] {  grid[i][j] = rune(ciphertext[i\*cols+j])  }  }  // Create a map to keep track of the original column order  colOrder := make(map[int]int)  for i, c := range key {  colOrder[i] = strings.IndexRune(key, c)  }  // Sort the column order by its values  sortedOrder := make([]int, cols)  for i := range sortedOrder {  sortedOrder[i] = colOrder[i]  }  for i := 0; i < cols; i++ {  for j := i + 1; j < cols; j++ {  if sortedOrder[i] > sortedOrder[j] {  sortedOrder[i], sortedOrder[j] = sortedOrder[j], sortedOrder[i]  }  }  }  sortedGrid := make([][]rune, rows)  for i := range sortedGrid {  sortedGrid[i] = make([]rune, cols)  for j := range sortedGrid[i] {  sortedGrid[i][colOrder[sortedOrder[j]]] = grid[i][j]  }  }  plaintext := ""  for i := 0; i < rows; i++ {  for j := 0; j < cols; j++ {  plaintext += string(sortedGrid[i][j])  }  }  // Remove any padding from the plaintext  plaintext = strings.TrimRight(plaintext, "X")  return plaintext  } |

Результат выполнения:





