Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Ульяновский государственный технический университет»

Кафедра «Вычислительная техника»

Дисциплина «Высокопроизводительные вычисления»

**Лабораторная работа № 4**

**Вариант 10**

**Блочные шифры, режимы использования блочных шифров**

Выполнил:

Студент группы ИВТАСбд-42

Сулейманов М.З.

Проверил:

Мартынов А.И.

Ульяновск

2024

**Цель работы**

Научиться реализовывать простейшие блочные алгоритмы шифрования и режимы их использования и строить на их основе более сложные блочные шифры.

**Задание**

Реализовать приложение с графическим интерфейсом, позволяющее выполнять следующие действия:

1. Шифровать и дешифровать текстовые и двоичные файлы с помощью блочного шифра

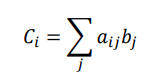
2. Сохранять зашифрованные/дешифрованные данные в файл

3. Загружать зашифрованные/дешифрованные данные из файла

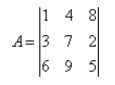
4. Хешировать текстовый пароль, который используется при шифровании для инициализации генератора псевдослучайных чисел, используемого в блочных шифрах, с помощью функции хеширования, реализованной в предыдущей лабораторной работе.

**Матричное шифрование**

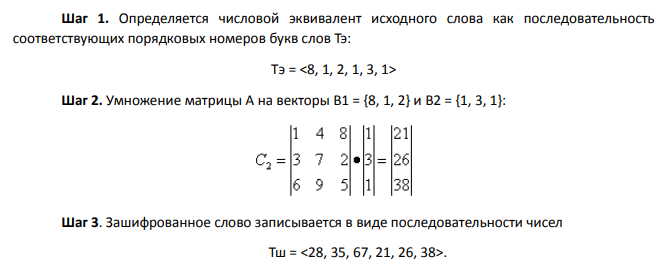
Для шифрования информации могут использоваться матричные преобразования. Наибольшее распространение получили методы шифрования, основанные на использовании матричной алгебры. Зашифрование k-го блока исходной информации, представленного в виде вектора Bk = |bj|, осуществляется путем перемножения матрицы-ключа А = |aij| и вектора Bk. В результате перемножения получается блок шифртекста в виде вектора Ck = |ci| , где элементы вектора Ck определяются по формуле:



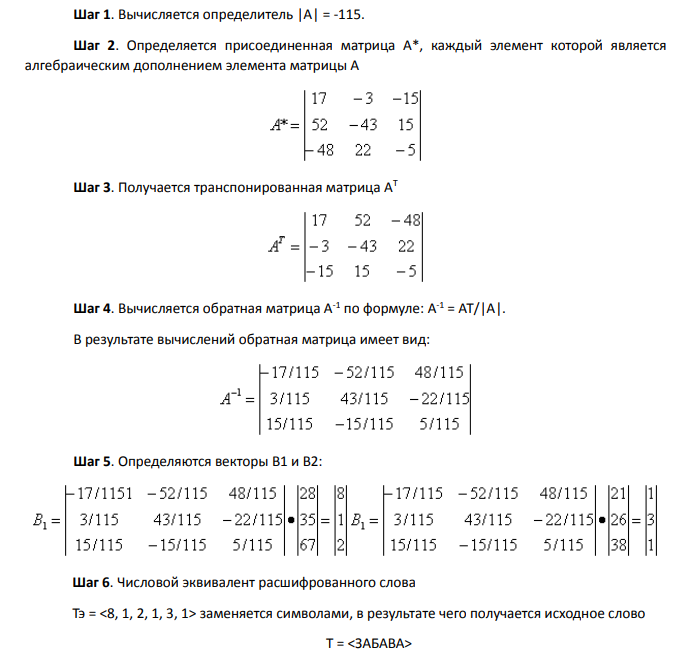
Расшифрование информации осуществляется путем последовательного перемножения векторов Ck и матрицы A-1, обратной матрице A.Пример шифрования информации с использованием алгебры матриц. Пусть необходимо зашифровать и расшифровать слово Т = < ЗАБАВА> с помощью матрицы-ключа А (ключ матрицы задается для каждого блока с использованием генератора псевдослучайных чисел):



Для шифрования исходного слова необходимо выполнить следующие шаги.



Дешифрование слова осуществляется следующим образом.



**Листинг**

|  |
| --- |
| import tkinter as tk  from tkinter import filedialog, ttk, messagebox  import numpy as np  import hashlib  import sympy as sp  modu = 53  symbols = " абвгдеёжзийклмнопрстуфхцчшщъыьэюя,:.?!\*'0123456789()"  chars = dict(enumerate(symbols))  def find\_key(char, data):      for key, value in data.items():          if(value==char):              return int(key)      return None  def park\_miller(seed, a=16807, m=2147483647):      """      Генератор псевдослучайных чисел Парка-Миллера.      Args:          seed: Целое число, начальное значение (seed).  Должно быть положительным.          a: Мультипликатор (по умолчанию 16807).          m: Модуль (по умолчанию 2147483647).      Yields:          Последовательность псевдослучайных чисел в диапазоне [0, 1).      """      if seed <= 0:          raise ValueError("Приближение должно быть положительным")      x = seed      while True:          x = (a \* x) % m          random\_number = x %10          yield random\_number  # --- Упрощенная реализация генератора случайных чисел и матричных операций ---  def generate\_key\_matrix(password):      """Генерирует матрицу ключа (упрощенная версия)."""      seed = int(hashlib.sha256(password.encode()).hexdigest(), 16)  # Хеширование пароля      seed = seed%(2\*\*32)      generator = park\_miller(seed)      arr = np.empty((7,7), dtype=int)      random\_numbers = [next(generator) for \_ in range(49)]      arr = np.empty((7,7), dtype=int)      for i in range(7):          for j in range(7):              arr[i][j]=random\_numbers[i\*j+j]        return arr  def invert\_matrix(matrix):      """Вычисляет обратную матрицу (упрощенная версия)."""      global modu      # Преобразование NumPy матрицы в sympy матрицу      sympy\_matrix = sp.Matrix(matrix.tolist())      try:          # Вычисление обратной матрицы в кольце вычетов          inverse\_matrix = sympy\_matrix.inv\_mod(modu)          # Преобразование обратно в NumPy матрицу          numpy\_inverse = np.array(inverse\_matrix.tolist(), dtype=int)          return numpy\_inverse      except sp.matrices.MatrixError:  #Обработка случая, когда матрица необратима          print("oh no")          return None  def matrix\_multiply(matrix, vector):      global modu      """Перемножает матрицу и вектор."""      return np.dot(matrix, vector)%modu  # --- Функции шифрования/дешифрования ---  def encrypt\_file(input\_filename, output\_filename, key\_matrix, password):      """Шифрует файл."""      global modu      try:          with open(input\_filename, "r", encoding="utf-8") as infile, open(output\_filename, "w", encoding="utf-8") as outfile:              while True:                  block = infile.read(7)                  if not block or block=='':                      break                  password = int(hashlib.sha256(password.encode()).hexdigest(), 16)  # Хеширование пароля                  password = str(password%(2\*\*32))                  key\_matrix = generate\_key\_matrix(password)                  vector = np.zeros(7, dtype=int) # инициализация нулями!                  for i in range(len(block)):                      if find\_key(block[i], chars)!=None:                          vector[i] = find\_key(block[i], chars)                  print(vector)                  encrypted\_vector = matrix\_multiply(key\_matrix, vector)                  for i in range(7):                      encrypted\_vector[i]=encrypted\_vector[i]%modu                      outfile.write(chars.get(encrypted\_vector[i]))        except Exception as e:          messagebox.showerror("Ошибка", f"Ошибка шифрования: {e}")  def decrypt\_file(input\_filename, output\_filename, key\_matrix, password):      """Дешифрует файл."""      global modu      try:          with open(input\_filename, "r", encoding="utf-8") as infile, open(output\_filename, "w", encoding="utf-8") as outfile:              while True:                  block = infile.read(7)                    if not block or block=='':                      break                  password = int(hashlib.sha256(password.encode()).hexdigest(), 16)  # Хеширование пароля                  password = str(password%(2\*\*32))                  key\_matrix = invert\_matrix(generate\_key\_matrix(password))                  vector = np.zeros(7, dtype=int) # инициализация нулями!                  for i in range(len(block)):                      if find\_key(block[i], chars)!=None:                          vector[i] = find\_key(block[i], chars)                  encrypted\_vector = matrix\_multiply(key\_matrix, vector)                  print(encrypted\_vector)                  for i in range(len(block)):                      encrypted\_vector[i]=encrypted\_vector[i]%modu                      outfile.write(chars.get(encrypted\_vector[i]))        except Exception as e:          messagebox.showerror("Ошибка", f"Ошибка дешифрования: {e}")    # --- Графический интерфейс ---  class MatrixCipherApp:      def \_\_init\_\_(self, master):  # Correct \_\_init\_\_ method          self.master = master          master.title("Матричное шифрование")          # ... (rest of your GUI code) ...          self.input\_file\_button = tk.Button(master, text="Выбрать входной файл", command=self.select\_input\_file)          self.input\_file\_button.pack()          self.input\_file\_entry = tk.Entry(master, width=50)          self.input\_file\_entry.pack()          self.output\_file\_button = tk.Button(master, text="Выбрать выходной файл", command=self.select\_output\_file)          self.output\_file\_button.pack()          self.output\_file\_entry = tk.Entry(master, width=50)          self.output\_file\_entry.pack()          self.password\_label = tk.Label(master, text="Пароль:")          self.password\_label.pack()          self.password\_entry = tk.Entry(master, show="\*", width=30)          self.password\_entry.pack()          self.encrypt\_button = tk.Button(master, text="Шифровать", command=self.encrypt)          self.encrypt\_button.pack()          self.decrypt\_button = tk.Button(master, text="Дешифровать", command=self.decrypt)          self.decrypt\_button.pack()          self.progress\_bar = ttk.Progressbar(master, orient="horizontal", length=200, mode="indeterminate")          self.progress\_bar.pack()          self.progress\_bar.pack\_forget()      def select\_input\_file(self):          filename = filedialog.askopenfilename()          self.input\_file\_entry.delete(0, tk.END)          self.input\_file\_entry.insert(0, filename)      def select\_output\_file(self):          filename = filedialog.asksaveasfilename()          self.output\_file\_entry.delete(0, tk.END)          self.output\_file\_entry.insert(0, filename)      def encrypt(self):          self.run\_cipher("encrypt")      def decrypt(self):          self.run\_cipher("decrypt")      def run\_cipher(self, mode):        input\_filename = self.input\_file\_entry.get()        output\_filename = self.output\_file\_entry.get()        password = self.password\_entry.get()        if not input\_filename or not output\_filename or not password:            messagebox.showerror("Ошибка", "Заполните все поля")            return        key\_matrix = generate\_key\_matrix(password)          self.progress\_bar.pack()        self.progress\_bar.start()        if mode == "encrypt":            encrypt\_file(input\_filename, output\_filename, key\_matrix, password)        elif mode == "decrypt":            decrypt\_file(input\_filename, output\_filename, key\_matrix, password)        self.progress\_bar.stop()        self.progress\_bar.pack\_forget()        messagebox.showinfo("Успех", "Операция завершена")  root = tk.Tk()  app = MatrixCipherApp(root)  root.mainloop() |