Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Ульяновский государственный технический университет»

Кафедра «Вычислительная техника»

Дисциплина «Информационная безопасность»

**Лабораторная работа №4**

«Блочные шифры, режимы использования блочных шифров»

Выполнил студент

группы ИВТАПбд-41:

Галацков И.А.

Проверил:

Мартынов А. И.

Ульяновск, 2024

**Задание на лабораторную работу**

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.

**Дополнительные требования к приложению:**

* Программа должна быть оформлена в виде удобной утилиты
* Программа должна обрабатывать файлы любого размера и содержания
* Текст программы оформляется прилично (удобочитаемо, с описанием ВСЕХ функций, переменных и критических мест).
* В процессе работы программа ОБЯЗАТЕЛЬНО выдает информацию о состоянии процесса шифрования / дешифрования.
* Интерфейс программы может быть произвольным, но удобным и понятным (разрешается использование библиотек GUI)
* Среда разработки и язык программирования могут быть произвольными.

**Алгоритм перестановки**

В алгоритме перестановки в каждом блоке меняется последовательность некоторых подблоков внутри блока, например байт или бит в слове, причем порядок перестановок определяется ключом.

*Листинг 1. Алгоритм перестановки*

|  |
| --- |
| def encrypt\_permutation(input\_file, output\_file, seed):  with open(input\_file, 'rb') as f:  data = f.read()   block\_size = 6  blocks = [data[i:i + block\_size] for i in range(0, len(data), block\_size)]   # Дополняем последний блок до 6 байт, если он короче  if len(blocks[-1]) < block\_size:  blocks[-1] = blocks[-1].ljust(block\_size, b'\x00')   encrypted\_blocks = []  for index, block in enumerate(blocks):  # Генерация m и n с привязкой к блоку  m, n = get\_m\_n(seed + index)   # Разделяем блок на три подблока  block1 = block[:m + 1]  block2 = block[m + 1:n + 1]  block3 = block[n + 1:]   # Перестановка: третий -> первый, второй -> второй, первый -> третий  encrypted\_block = block3 + block2 + block1  encrypted\_blocks.append(encrypted\_block)   with open(output\_file, 'wb') as f:  f.write(b''.join(encrypted\_blocks))  def decrypt\_permutation(input\_file, output\_file, seed):  with open(input\_file, 'rb') as f:  data = f.read()   block\_size = 6  blocks = [data[i:i + block\_size] for i in range(0, len(data), block\_size)]   decrypted\_blocks = []  for index, block in enumerate(blocks):  # Генерация m и n с привязкой к блоку  m, n = get\_m\_n(seed + index)   # Вычисляем размеры подблоков  len\_block1 = m + 1  len\_block2 = n - m  len\_block3 = block\_size - len\_block1 - len\_block2   # Разделяем зашифрованный блок на три подблока  block3 = block[:len\_block3]  block2 = block[len\_block3:len\_block3 + len\_block2]  block1 = block[len\_block3 + len\_block2:]   # Перестановка: первый -> третий, второй -> второй, третий -> первый  decrypted\_block = block1 + block2 + block3  decrypted\_blocks.append(decrypted\_block)   # Убираем дополнение нулями из последнего блока  with open(output\_file, 'wb') as f:  f.write(b''.join(decrypted\_blocks).rstrip(b'\x00')) |

Функция выполняет перестановочное шифрование данных из файла. Она разбивает данные на блоки фиксированной длины, производит перестановку частей каждого блока на основе параметра seed, а затем записывает зашифрованные данные в выходной файл. Вот как это работает:

1. **Чтение входного файла**  
   Функция открывает входной файл (input\_file) в бинарном режиме и считывает его содержимое в переменную data.
2. **Разделение данных на блоки**  
   Данные разбиваются на блоки длиной 6 байт. Если последний блок меньше 6 байт, он дополняется нулями (\x00), чтобы все блоки имели одинаковую длину.
3. **Шифрование блоков**  
   Для каждого блока выполняется следующее:
   * Генерируются значения m и n на основе начального параметра seed и текущего индекса блока. Это определяет, как блок будет разделён на три части. Функция get\_m\_n должна возвращать два индекса, которые зависят от seed и текущего блока.
   * Блок разделяется на три части:
     + Первая часть (block1) включает байты от начала блока до индекса m.
     + Вторая часть (block2) включает байты между m + 1 и n.
     + Третья часть (block3) включает оставшиеся байты от n + 1 до конца блока.
   * Перестановка частей: третья часть становится первой, вторая остаётся на месте, а первая становится последней. Это создаёт зашифрованный блок.
   * Зашифрованный блок добавляется в список.
4. **Сборка зашифрованных данных**  
   Все зашифрованные блоки объединяются в один массив байтов.
5. **Запись в выходной файл**  
   Результирующие данные записываются в выходной файл (output\_file) в бинарном режиме.

Функция дешифрования работает аналогичным образом.

**Вывод**

В данной лабораторной работе был реализован алгоритм перестановки, позволяющий шифровать и дешифровать данные.