Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Пермский национальный исследовательский политехнический университет»

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

Направление 09.03.04 – «Программная инженерия»

Профиль: «Разработка программно-информационных систем»

Дисциплина: «Защита информации»

Семестр 5

ОТЧЕТ

по лабораторной работе №4

Тема: «Блочное шифрование методом DES»

Вариант 19

|  |  |  |
| --- | --- | --- |
|  | Выполнил: Студент группы РИС-23-2б  Д.Е. Колосов | |
|  |  | подпись, дата |
|  | Проверил: Ст. преподаватель кафедры ИТАС  В. Г. Шереметьев | |
|  |  | подпись, дата |

Пермь, 2025

**Цель работы**

Получить практические навыки по использованию блочных алгоритмов шифрования, на примере использования алгоритма DES (Data Encryption Standard).

**Задание**

Реализовать шифрование бинарного файла, методом блочного шифрования, используя блоки длиной 16 бит, ключ длиной 16 бит, реализуя в алгоритме шифрования методику DES.

**ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Основные понятия блочного шифрования:

Блоки - файл разбивается на кусочки фиксированного размера (в оригинальном DES - 64 бита, у нас в программе - 16 бит)

Ключ - секретный набор цифр для шифрования/расшифровки.

Как работает DES.

Подготовка:

Файл разбивается на блоки (например, по 16 бит = 2 байта)

Генерируем ключи - из основного ключа создаём несколько "раундовых ключей"

Для каждого блока:

Начальная перестановка - перемешиваем биты в блоке по определённому правилу.

Блок: 1101 (13 в десятичной, 4 бита)

Таблица перестановки: [3, 1, 0, 2] (переставляем биты)

Размер блока: 4 бита (для простоты)

Позиции битов:

Позиции: 3 2 1 0

Биты: 1 1 0 1

Значение: 8 4 2 1

Пошаговый процесс:

Шаг 1: i=0, pos=3

Смотрим на позицию 3 исходного блока: 1 (бит установлен)

Устанавливаем бит в позиции 0 результата:

Результат: 1000 (8 в десятичной)

Шаг 2: i=1, pos=1

Смотрим на позицию 1 исходного блока: 0 (бит НЕ установлен)

Ничего не делаем:

Результат: 1000 (без изменений)

Шаг 3: i=2, pos=0

Смотрим на позицию 0 исходного блока: 1 (бит установлен)

Устанавливаем бит в позиции 2 результата:

Результат: 1100 (12 в десятичной)

Шаг 4: i=3, pos=2

Смотрим на позицию 2 исходного блока: 1 (бит установлен)

Устанавливаем бит в позиции 3 результата:

Результат: 1110 (14 в десятичной)

Итог: 1101 → 1110

Раунды фейстеля (основная магия) - повторяем несколько раз:

Делим блок на левую и правую половинки.

Правая половина проходит через функцию Фейстеля:

Расширение - увеличиваем размер с 8 до 16 бит.

Пример: 8 бит → 16 бит

Исходный блок (8 бит): 10110010

Позиции битов (8 бит):

7 6 5 4 3 2 1 0

1 0 1 1 0 0 1 0

Таблица расширения: [0, 1, 2, 3, 2, 3, 4, 5, 4, 5, 6, 7, 6, 7, 0, 1]

Что делает таблица:

Бит 0 результата = бит 0 исходного

Бит 1 результата = бит 1 исходного

Бит 2 результата = бит 2 исходного

Бит 3 результата = бит 3 исходного

Бит 4 результата = бит 2 исходного

Бит 5 результата = бит 3 исходного

и т.д.

Результат (16 бит): 10110010 10110010

XOR с ключом - смешиваем с раундовым ключом (расширяли как раз для того, чтобы было одинаковое количество бит с ключом -16 и можно было сделать исключающее или).

S-блоки - главная нелинейная операция (табличная замена).

Пример:

S-бокс таблица:

s\_boxes = [

# Строка 0: [14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7]

# Строка 1: [0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8]

# и т.д.

]

Блок на входе: 1011001010110010 (16 бит)

Разбиваем на 4 группы по 4 бита:

Группа 1: 1011 = 11

Группа 2: 0010 = 2

Группа 3: 1011 = 11

Группа 4: 0010 = 2

Применяем S-бокс:

Группа 1: 11 → смотрим в таблицу: s\_boxes[0][0][11] = 12 → 1100

Группа 2: 2 → s\_boxes[0][1][2] = 7 → 0111

Группа 3: 11 → s\_boxes[0][2][11] = 11 → 1011

Группа 4: 2 → s\_boxes[0][3][2] = 2 → 0010

Результат: 1100011110110010

P-БЛОК ПЕРЕСТАНОВКА

Перестановка - снова перемешиваем биты по таблице перестановки.

Пример с 4 битами (упрощённо):

Таблица: [1, 3, 0, 2]

Исходный блок: 1100 (12 в десятичной)

Позиции:

3 2 1 0

1 1 0 0

Перестановка:

Бит 0 результата = бит 1 исходного = 1

Бит 1 результата = бит 3 исходного = 1

Бит 2 результата = бит 0 исходного = 0

Бит 3 результата = бит 2 исходного = 1

Результат: 1101 (13 в десятичной)

XOR с левой половиной - результат функции XOR'им с левой половиной.

Пусть перед раундом:

Левая половина (L): 1100 (12)

Правая половина (R): 1010 (10)

Допустим, функция Фейстеля вернула: 0111 (7)

Шаг 1: XOR с левой половиной

Левая: 1100

XOR Фейстель: 0111

Результат: 1011 ← это новая правая половина

Меняем половинки местами.

Старая левая становится новой правой: 1100 → правая

Результат XOR становится новой левой: 1011 → левая

Результат раунда:

Левая: 1011 (11)

Правая: 1100 (12)

Финальная перестановка - обратная начальной.

Начальная перестановка:

initial\_perm = [2, 6, 10, 14, 3, 7, 11, 15, 1, 5, 9, 13, 0, 4, 8, 12]

Финальная перестановка:

final\_perm = [11, 7, 3, 15, 10, 6, 2, 14, 9, 5, 1, 13, 8, 4, 0, 12]

Пример как находить обратную:

Прямая перестановка: [2, 0, 3, 1]

Позиция 0 → идёт в позицию 2

Позиция 1 → идёт в позицию 0

Позиция 2 → идёт в позицию 3

Позиция 3 → идёт в позицию 1

Чтобы найти обратную:

Куда пошла позиция 0? В позицию 2 → значит в обратной: позиция 2 → позиция 0

Куда пошла позиция 1? В позицию 0 → значит в обратной: позиция 0 → позиция 1

Куда пошла позиция 2? В позицию 3 → значит в обратной: позиция 3 → позиция 2

Куда пошла позиция 3? В позицию 1 → значит в обратной: позиция 1 → позиция 3

Обратная: [1, 3, 0, 2]

Расшифровка - всё то же самое, но:

Раундовые ключи в обратном порядке

# Шифрование: ключи K1, K2, K3, K4

# Расшифровка: ключи K4, K3, K2, K1

Гарантия одинаковых сгенерированных K1, K2, K3, K4 заключается в использовании одного главного ключа для шифрования и для расшифровки.

**ХОД РАБОТЫ**

Главное окно программы представлено на рисунке 1.

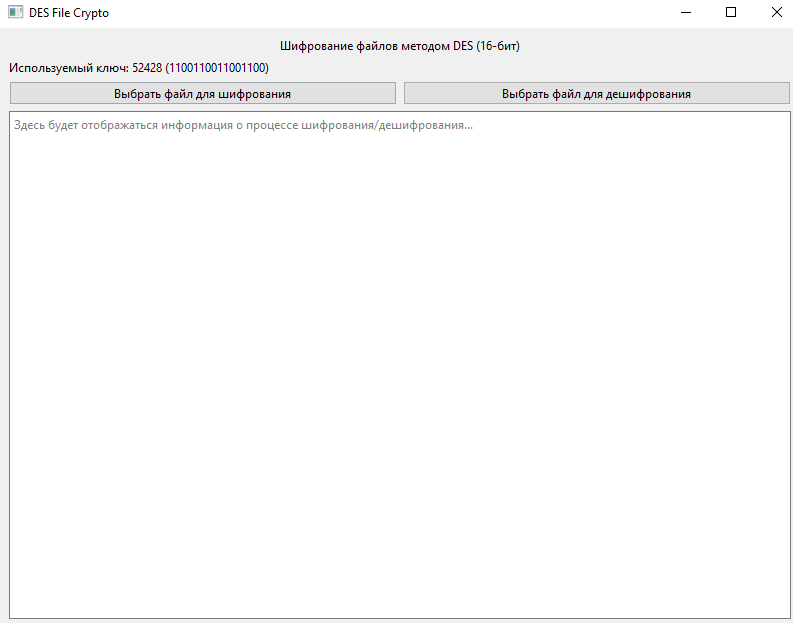


Рисунок 1 – главное окно программы

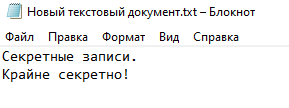


Рисунок 2 – содержание файла, который будем шифровать

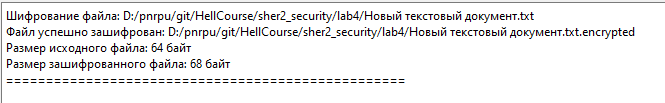


Рисунок 3 – шифрование файла

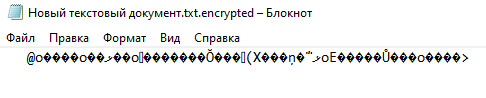


Рисунок 4 – содержимое зашифрованного файла

Используя тот же ключ, можно выполнить расшифровку данных.

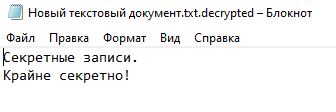


Рисунок 5 – расшифрование сообщения

**ПРИЛОЖЕНИЕ А**

**Листинг программы**

import sys

import os

from PySide6.QtWidgets import (QApplication, QMainWindow, QVBoxLayout,

QHBoxLayout, QPushButton, QTextEdit,

QLabel, QFileDialog, QWidget, QMessageBox)

from PySide6.QtCore import Qt

class SimpleDES:

"""

Упрощенная реализация DES для 16-битных блоков

"""

def \_\_init\_\_(self, key):

# Преобразуем ключ в 16-битное число

self.key = key & 0xFFFF

# Упрощенные S-блоки (для учебных целей)

self.s\_boxes = [

# S-box 1

[

[14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7],

[0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8],

[4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0],

[15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13]

]

]

def permute(self, block, permutation):

"""

Выполняет перестановку битов в блоке согласно таблице permutation

"""

result = 0

for i, pos in enumerate(permutation):

# Если бит в позиции pos установлен, устанавливаем соответствующий бит в результате

if block & (1 << (15 - pos)):

result |= (1 << (15 - i))

"""

Переставляет биты согласно таблице permutation

permutation = [2, 0, 1] # таблица перестановки

block = 0b110 (6 в десятичной)

i=0, pos=2: проверяем 2-й бит исходного блока (1), ставим в 0-ю позицию результата

i=1, pos=0: проверяем 0-й бит исходного блока (0), ставим в 1-ю позицию результата

i=2, pos=1: проверяем 1-й бит исходного блока (1), ставим в 2-ю позицию результата

Результат: 0b101 (5 в десятичной)"""

return result

def expand(self, block):

"""

Расширяет 8-битный блок до 16 бит для XOR с ключом

"""

# Простая расширяющая перестановка

expansion\_table = [3, 0, 1, 2, 1, 2, 3, 0, 3, 0, 1, 2, 1, 2, 3, 0]

return self.permute(block, expansion\_table)

def s\_box\_substitution(self, block):

"""

Применяет S-блоки к блоку

"""

result = 0

# Разбиваем 16-битный блок на 4 части по 4 бита

for i in range(4):

# Извлекаем 4 бита

part = (block >> (12 - i \* 4)) & 0xF

# Применяем S-бокс

row = ((part >> 3) & 0x1) \* 2 + ((part >> 2) & 0x1)

col = part & 0x3

sbox\_value = self.s\_boxes[0][row][col]

result = (result << 4) | sbox\_value

return result

def p\_box\_permutation(self, block):

"""

P-бокс перестановка

"""

p\_box = [1, 5, 9, 13, 2, 6, 10, 14, 3, 7, 11, 15, 4, 8, 12, 0]

return self.permute(block, p\_box)

def feistel\_function(self, right\_half, round\_key):

"""

Функция Фейстеля - сердце алгоритма DES

"""

# 1. Расширяем правую половину (8 бит -> 16 бит)

expanded = self.expand(right\_half)

# 2. XOR с раундовым ключом

xored = expanded ^ round\_key

# 3. S-бокс подстановка

substituted = self.s\_box\_substitution(xored)

# 4. P-бокс перестановка

return self.p\_box\_permutation(substituted)

def generate\_round\_keys(self):

"""

Генерирует раундовые ключи

"""

keys = []

key = self.key

for i in range(4): # 4 раунда

# Циклический сдвиг влево на 3 бита

key = ((key << 3) | (key >> 13)) & 0xFFFF

"""

key << 3 - сдвигаем все биты влево на 3 позиции

key >> 13 - сдвигаем все биты вправо на 13 позиций

| - побитовое ИЛИ (объединяем результаты)

& 0xFFFF - обрезаем до 16 бит (маска)"""

keys.append(key)

return keys

def encrypt\_block(self, block):

"""

Шифрует один 16-битный блок

"""

# Генерируем раундовые ключи

round\_keys = self.generate\_round\_keys()

# Начальная перестановка

initial\_perm = [1, 5, 9, 13, 2, 6, 10, 14, 3, 7, 11, 15, 4, 8, 12, 0]

block = self.permute(block, initial\_perm)

# Разделяем блок на левую и правую части (по 8 бит)

left = (block >> 8) & 0xFF

right = block & 0xFF

# 4 раунда Фейстеля

for i in range(4):

# Сохраняем правую часть

temp = right

# Правая часть = левая часть XOR feistel(правая часть, ключ)

right = left ^ self.feistel\_function(temp, round\_keys[i])

# Левая часть = старая правая часть

left = temp

# Объединяем части (после последнего раунда не меняем местами)

result = (left << 8) | right

# Конечная перестановка (обратная начальной)

final\_perm = [15, 11, 7, 3, 14, 10, 6, 2, 13, 9, 5, 1, 12, 8, 4, 0]

return self.permute(result, final\_perm)

def decrypt\_block(self, block):

"""

Дешифрует один 16-битный блок

"""

# Генерируем раундовые ключи

round\_keys = self.generate\_round\_keys()

# Начальная перестановка (такая же как при шифровании)

initial\_perm = [1, 5, 9, 13, 2, 6, 10, 14, 3, 7, 11, 15, 4, 8, 12, 0]

block = self.permute(block, initial\_perm)

# Разделяем блок на левую и правую части

left = (block >> 8) & 0xFF

right = block & 0xFF

# 4 раунда Фейстеля в обратном порядке

for i in range(3, -1, -1): # от 3 до 0

# Сохраняем левую часть

temp = left

# Левая часть = правая часть XOR feistel(левая часть, ключ)

left = right ^ self.feistel\_function(temp, round\_keys[i])

# Правая часть = старая левая часть

right = temp

# Объединяем части

result = (left << 8) | right

# Конечная перестановка

final\_perm = [15, 11, 7, 3, 14, 10, 6, 2, 13, 9, 5, 1, 12, 8, 4, 0]

return self.permute(result, final\_perm)

class CryptoApp(QMainWindow):

"""

Главное окно приложения для шифрования/дешифрования файлов

"""

def \_\_init\_\_(self):

super().\_\_init\_\_()

# Устанавливаем ключ шифрования (16 бит)

self.key = 0b1100110011001100 # Пример ключа

# Инициализируем DES

self.des = SimpleDES(self.key)

# Настраиваем интерфейс

self.init\_ui()

def init\_ui(self):

"""Инициализация пользовательского интерфейса"""

# Создаем центральный виджет

central\_widget = QWidget()

self.setCentralWidget(central\_widget)

# Создаем основной layout

layout = QVBoxLayout()

central\_widget.setLayout(layout)

# Заголовок

title = QLabel("Шифрование файлов методом DES (16-бит)")

title.setAlignment(Qt.AlignCenter)

layout.addWidget(title)

# Отображение ключа

key\_label = QLabel(f"Используемый ключ: {self.key} ({bin(self.key)[2:]})") # cо 2 чтоб 0b не было видно - призник битовой записи числа

layout.addWidget(key\_label)

# Кнопки для выбора файлов

file\_buttons\_layout = QHBoxLayout()

self.encrypt\_btn = QPushButton("Выбрать файл для шифрования")

self.encrypt\_btn.clicked.connect(self.encrypt\_file)

file\_buttons\_layout.addWidget(self.encrypt\_btn)

self.decrypt\_btn = QPushButton("Выбрать файл для дешифрования")

self.decrypt\_btn.clicked.connect(self.decrypt\_file)

file\_buttons\_layout.addWidget(self.decrypt\_btn)

layout.addLayout(file\_buttons\_layout)

# Текстовое поле для вывода информации

self.text\_output = QTextEdit()

self.text\_output.setPlaceholderText("Здесь будет отображаться информация о процессе шифрования/дешифрования...")

layout.addWidget(self.text\_output)

# Настраиваем главное окно

self.setWindowTitle("DES File Crypto")

self.setGeometry(100, 100, 800, 600)

def log\_message(self, message):

"""Добавляет сообщение в текстовое поле"""

self.text\_output.append(message)

def encrypt\_file(self):

"""Шифрует выбранный файл"""

try:

# Открываем диалог выбора файла

file\_path, \_ = QFileDialog.getOpenFileName(self, "Выберите файл для шифрования")

if not file\_path:

return

self.log\_message(f"Шифрование файла: {file\_path}")

# Генерируем имя для зашифрованного файла

encrypted\_path = file\_path + ".encrypted"

# Получаем размер исходного файла

original\_size = os.path.getsize(file\_path)

# Читаем и шифруем файл

with open(file\_path, 'rb') as input\_file:

with open(encrypted\_path, 'wb') as output\_file:

# Записываем размер исходного файла в начало

output\_file.write(original\_size.to\_bytes(4, byteorder='big'))

while True:

# Читаем блок данных (2 байта = 16 бит)

chunk = input\_file.read(2)

if not chunk:

break

# Если блок меньше 2 байт, дополняем нулями

if len(chunk) < 2:

chunk = chunk + b'\x00' \* (2 - len(chunk))

# Преобразуем байты в 16-битное число

block = int.from\_bytes(chunk, byteorder='big')

# Шифруем блок

encrypted\_block = self.des.encrypt\_block(block)

# Записываем зашифрованный блок

output\_file.write(encrypted\_block.to\_bytes(2, byteorder='big'))

encrypted\_size = os.path.getsize(encrypted\_path)

self.log\_message(f"Файл успешно зашифрован: {encrypted\_path}")

self.log\_message(f"Размер исходного файла: {original\_size} байт")

self.log\_message(f"Размер зашифрованного файла: {encrypted\_size} байт")

self.log\_message("=" \* 50)

except Exception as e:

QMessageBox.critical(self, "Ошибка", f"Ошибка при шифровании: {str(e)}")

def decrypt\_file(self):

"""Дешифрует выбранный файл"""

try:

# Открываем диалог выбора файла

file\_path, \_ = QFileDialog.getOpenFileName(self, "Выберите файл для дешифрования")

if not file\_path:

return

self.log\_message(f"Дешифрование файла: {file\_path}")

# Генерируем имя для расшифрованного файла

if file\_path.endswith('.encrypted'):

decrypted\_path = file\_path[:-10] + ".decrypted"

else:

decrypted\_path = file\_path + ".decrypted"

# Читаем и дешифруем файл

with open(file\_path, 'rb') as input\_file:

# Читаем размер исходного файла

original\_size\_bytes = input\_file.read(4)

if len(original\_size\_bytes) < 4:

raise ValueError("Файл слишком короткий")

original\_size = int.from\_bytes(original\_size\_bytes, byteorder='big')

with open(decrypted\_path, 'wb') as output\_file:

bytes\_written = 0

while True:

# Читаем блок данных (2 байта = 16 бит)

chunk = input\_file.read(2)

if not chunk:

break

# Если блок меньше 2 байт, дополняем нулями

if len(chunk) < 2:

chunk = chunk + b'\x00' \* (2 - len(chunk))

# Преобразуем байты в 16-битное число

block = int.from\_bytes(chunk, byteorder='big')

# Дешифруем блок

decrypted\_block = self.des.decrypt\_block(block)

# Преобразуем обратно в байты

decrypted\_bytes = decrypted\_block.to\_bytes(2, byteorder='big')

# Убираем дополняющие нули в конце файла

if bytes\_written + 2 > original\_size:

bytes\_to\_write = original\_size - bytes\_written

decrypted\_bytes = decrypted\_bytes[:bytes\_to\_write]

output\_file.write(decrypted\_bytes)

bytes\_written += len(decrypted\_bytes)

if bytes\_written >= original\_size:

break

self.log\_message(f"Файл успешно расшифрован: {decrypted\_path}")

# Показываем содержимое если это текстовый файл

self.show\_file\_content(decrypted\_path)

self.log\_message("=" \* 50)

except Exception as e:

QMessageBox.critical(self, "Ошибка", f"Ошибка при дешифровании: {str(e)}")

def show\_file\_content(self, file\_path):

"""Показывает содержимое файла если это текст"""

try:

with open(file\_path, 'rb') as file:

content = file.read()

# Проверяем, является ли файл текстовым

try:

text\_content = content.decode('utf-8')

self.log\_message("Содержимое файла:")

self.log\_message(text\_content[:1000] + "..." if len(text\_content) > 1000 else text\_content)

except UnicodeDecodeError:

self.log\_message("Файл содержит бинарные данные (не текст)")

self.log\_message(f"Размер файла: {len(content)} байт")

except Exception as e:

self.log\_message(f"Не удалось прочитать содержимое файла: {str(e)}")

def main():

"""

Главная функция, запускает приложение

"""

# Создаем экземпляр приложения

app = QApplication(sys.argv)

# Создаем и показываем главное окно

window = CryptoApp()

window.show()

# Запускаем главный цикл приложения

sys.exit(app.exec())

# Точка входа в программу

if \_\_name\_\_ == "\_\_main\_\_":

main()