Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа № 5
по дисциплине «Защита информация»

Тема Реализация алгоритма симметричного шифрования (DES).

Студент Шахнович Дмитрий Сергеевич

Группа ИУ7-72Б

Преподаватель Руденкова Ю.С.

Москва, 2025

Содержание

ВВЕДЕНИЕ

Цель работы: Разработка алгоритма симметричного шифрования (DES). Шифрование и расшифровка произвольного файла.

Задачи:

1) проанализировать алгоритмы шифрования и расшифровки DES;
2) реализовать программу шифрования симметричным алгоритмом DES;
3) обеспечить шифрование и расшифровку произвольного файла с использованием разработанной программы.

1 Аналитическая часть

1.1 Определения

Блочный шифр – разновидность симметричного шифра, оперирующего группами бит фиксированной длины — блоками. Если исходный текст (или его остаток) меньше размера блока, перед шифрованием его дополняют. Для шифрования блока используется ключ и функция, которые не меняются между блоками.

Поточный шифр – это симметричный шифр, в котором каждый символ или байт открытого текста преобразуется в символ шифрованного текста в зависимости не только от используемого ключа, но и от его расположения в потоке открытого текста.

На рисунке 1.1 представлена схема блочного шифра, где

— M_i – блоки открытого текста;

— K – ключ шифрования;

— E_k – алгоритм шифрования, принимающий блок открытого текста и ключ;

— C_i – блок зашифрованного текста.

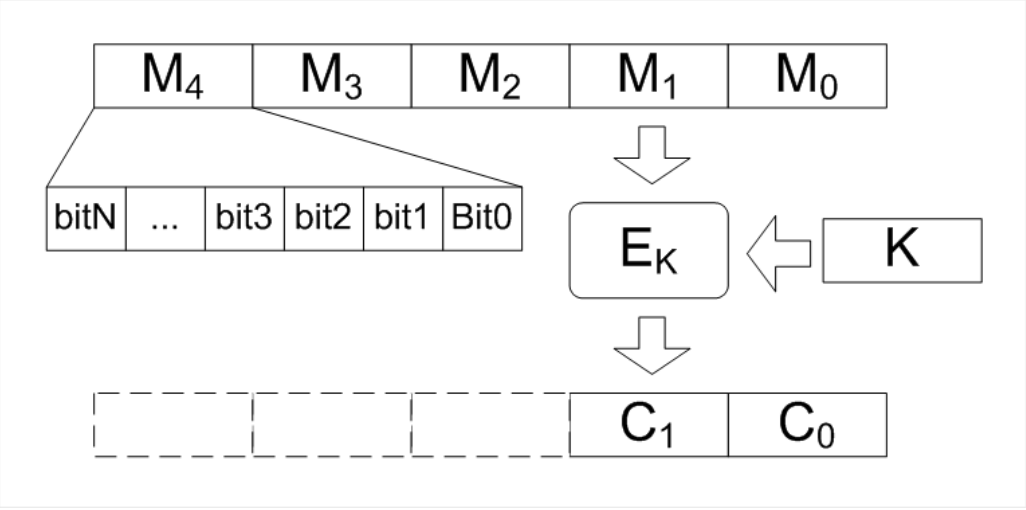

Рисунок 1.1 — Обобщённая схема блочного шифра

Алгоритм подстановки – заменяет каждый элемент исходного открытого текста (букву, байт, блок) на другой элемент, в соответствии с фиксированным правилом или таблицей. Пример: Шифр Цезаря.

Алгоритм перестановки – изменяет порядок следования элементов открытого текста, оставляя сами элементы неизменными. Пример: шифр простой перестановки, который переставляет местами символы, согласно таблице.

Пример алгоритма, использующего оба подхода: Сеть фейстеля – сначала переставляет биты исходного блока, а затем по нескольким таблицам заменяет блоки по 4 бита. Используется как часть алгоритма DES.

1.2 Алгоритм DES

DES – алгоритм для симметричного шифрования, разработанный фирмой IBM и утверждённый правительством США в 1977 году как официальный стандарт (FIPS 46-3). Размер блока для DES равен 64 битам. В основе алгоритма лежит сеть Фейстеля с 16 циклами (раундами) и ключом, имеющим длину 56 бит (дополняется до 64 бит для удобства передачи).

1.2.1 Генерация подключей

На основании ключа, перед началом алгоритма, генерируются 16 подключей – по одному на каждый цикл шифрования сетью фейстеля.

Алгоритм генерации ключей имеет следующий вид:

1) разделить исходный ключ на две части по 28 бит C_0 и D_0, беря отдельные биты из исходного ключа;
2) получить C_i и D_i для $i = 1..16$, последовательным циклическим сдвигом предыдущих частей на 1 или 2, согласно специальной таблице;
3) получить подключи k_i длиной 48 бит для $i = 1..16$ беря биты из C_iD_i согласно специальной таблице.

1.2.2 Функция фейстеля

Функция фейстеля – основная часть алгоритма DES, которая последовательно применяется 16 раз к блокам данных.

Функция принимает в качестве аргументов блок размером 32 бита и ключ размером 48 бит.

Алгоритм функции состоит из нескольких этапов и имеет вид:

— расширение исходного блока до 48 бит повторением отдельных битов исходного блока с помощью таблицы;

— побитовое сложение по модуля два (xor) расширенного блока и ключа;

— преобразование результата сложения из 48 бит в 32 по серии из 8 таблиц;

— конечная перестановка блока согласно таблице.

Результатом является новый блок размером 32 бита.

На рисунке 1.2 представлена схема функции фейстеля.

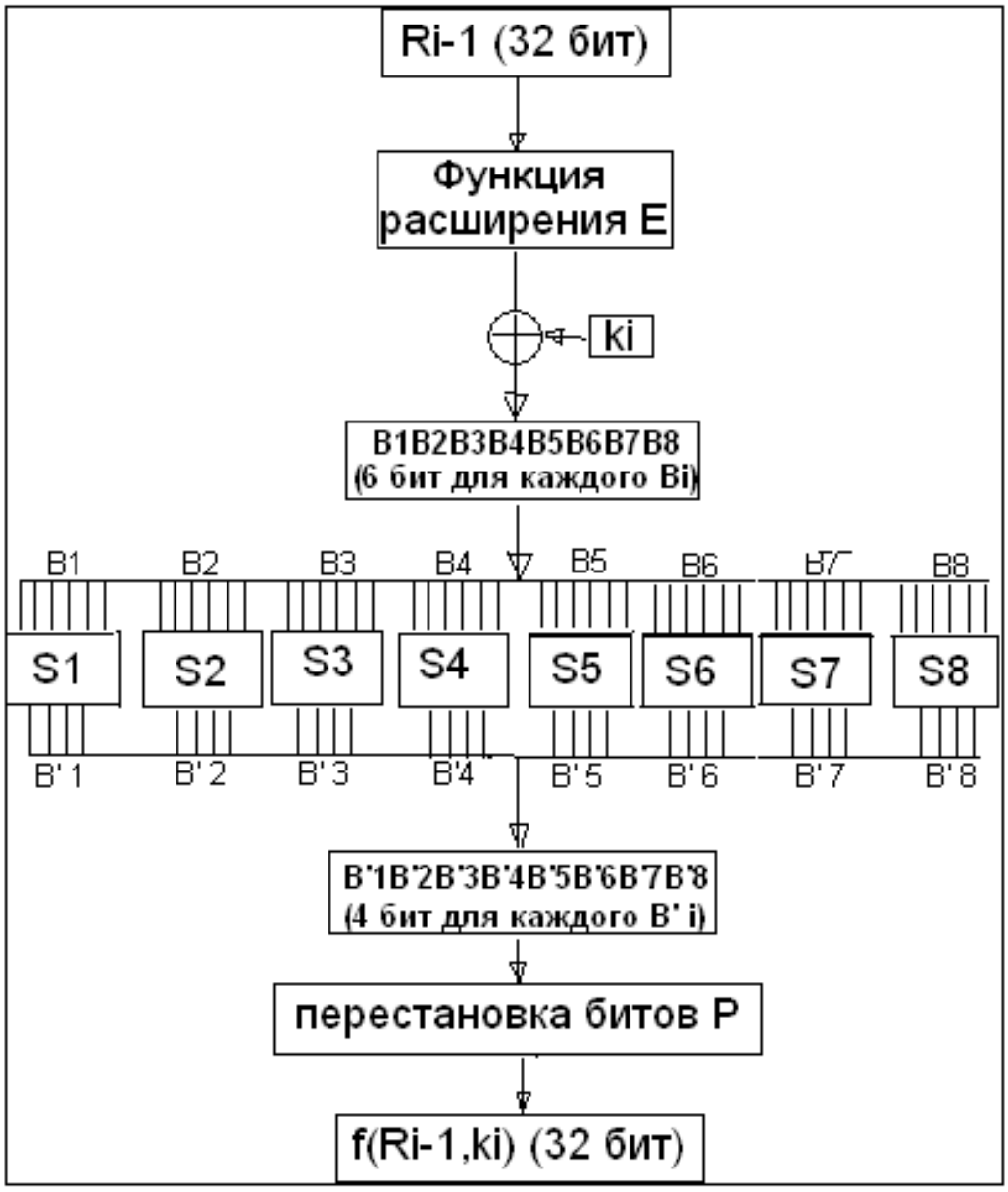

Рисунок 1.2 — Схема функции фейстеля

1.2.3 Алгоритм шифрования блока

Размер блока для шифрования алгоритмом DES – 64 бит

Алгоритм шифрования отдельного блока имеет следующий вид:

1) Начальная перестановка блока согласно таблице;

2) Разбить блок на две части L_0 и R_0 – старшие и младшие 32 бита соответственно;

3) Цикл по i от 1 до 16:

   3.1) L_i = R_{i-1};

   3.2) R_i = $L_i \; xor \; feistel(R_{i-1}, k_i)$

4) $T_{16} = R_{16}L_{16}$

5) Конечная перестановка блока T_{16} согласно таблице.

Результатом является блок размером 64 бита – зашифрованный блок.

На рисунке 1.3 представлена схема шифрования блока DES.

1.2.4 Алгоритм расшифровки блока

Алгоритм расшифровки отдельного блока очень похож на алгоритм шифрования и имеет следующий вид:

1) Начальная перестановка зашифрованного блока согласно таблице;

2) Разбить блок на две части L_0 и R_0 – старшие и младшие 32 бита соответственно;

3) Цикл по i от 1 до 16:

   3.1) R_i = L_{i-1};

   3.2) L_i = $R_i \; xor \; feistel(L_{i-1}, k_i)$

4) $T_{16} = R_{16}L_{16}$

5) Конечная перестановка блока T_{16} согласно таблице.

Результатом является блок размером 64 бита – блок открытого текста.

На рисунке 1.4 представлена схема расшифровки блока DES.

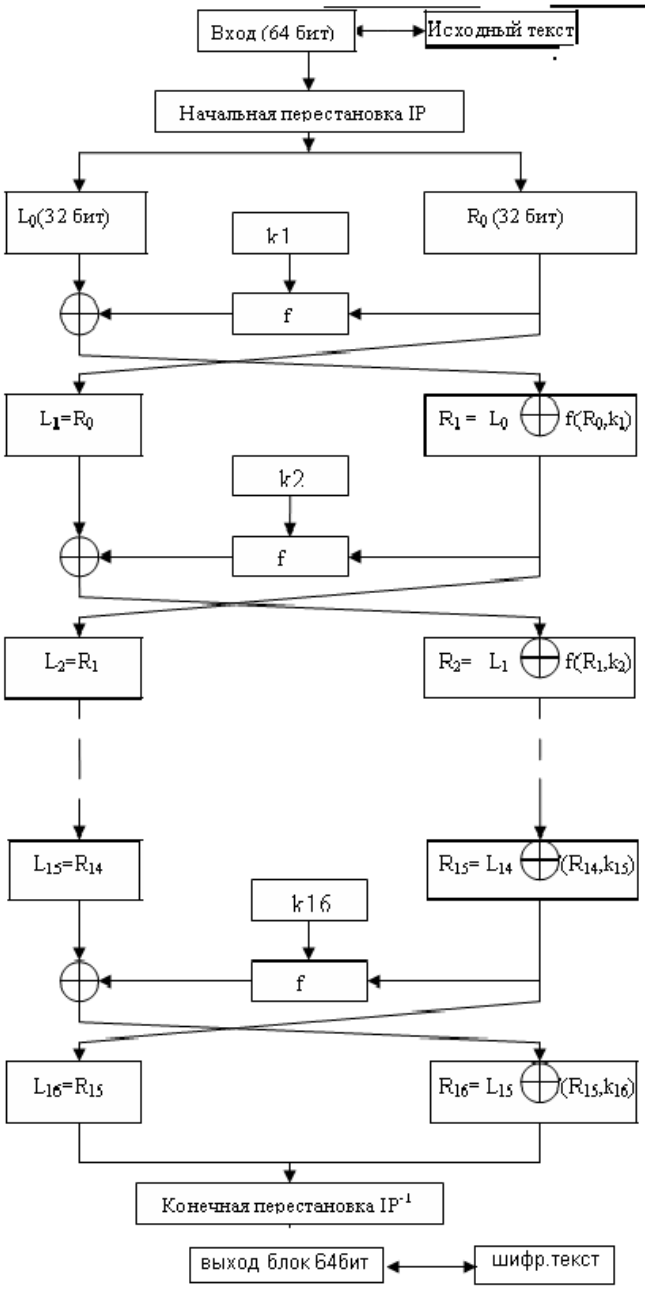

Рисунок 1.3 — Схема шифрования алгоритма DES

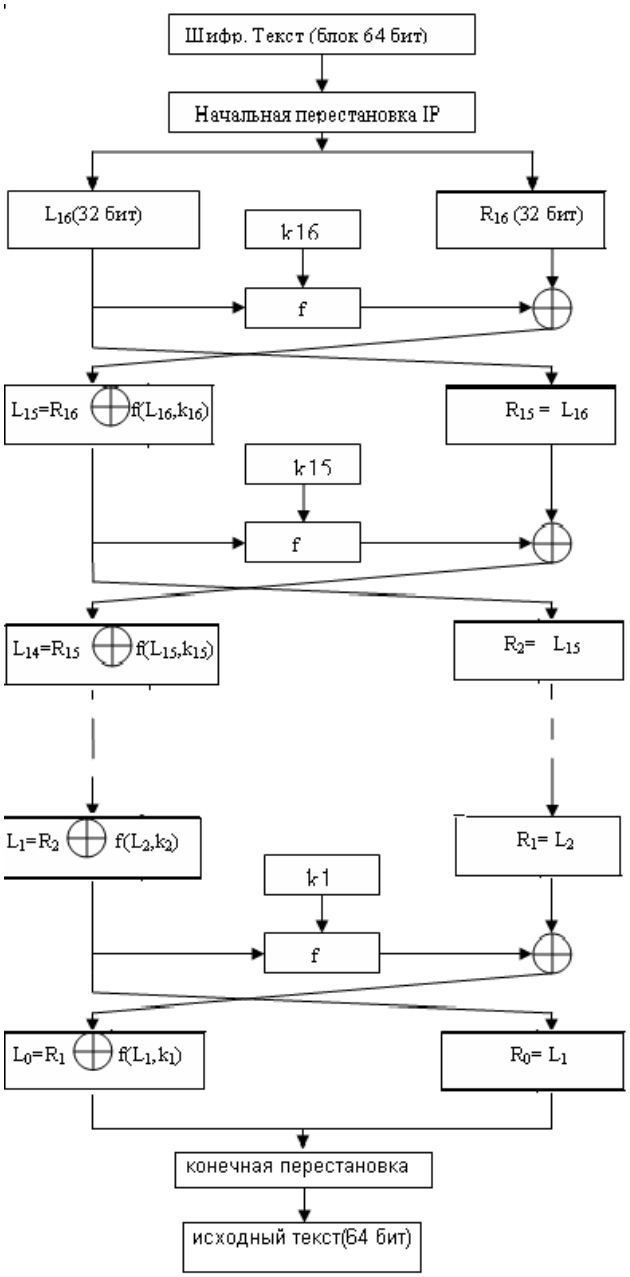

Рисунок 1.4 — Схема расшифровки алгоритма DES

2 Технологическая часть

2.1 Средства реализации

В качестве языка программирования для программной реализации алгоритма DES был выбран Go.

2.2 Код разработанной программы

На листингах 2.1- 2.4 представлена разработанная программа.

Листинг 2.1 — Код генерации ключей и подключей

```
package des

import (
  "crypto/rand"
  "encoding/binary"
  "fmt"
  "info-security/internal/utils/perm"
  "io"
)

type DESKey struct {
  Key uint64
}

func NewDESKey(key uint64) *DESKey {
  return &DESKey{
    Key: key,
  }
}

func RandomDESKey() *DESKey {
  keyBytes := make([]byte, 8)
  _, err := rand.Read(keyBytes)
  if err != nil {
    panic(err)
  }
  for i := 0; i < 8; i++ {
    keyBytes[i] = setParity(keyBytes[i])
  }
```

```
  return NewDESKey(binary.BigEndian.Uint64(keyBytes))
}

func setParity(b byte) byte {
  ones := 0
  for i := 0; i < 7; i++ {
    if (b & (1 << i)) != 0 {
      ones++
    }
  }
  // set parity bit so that the number of 1s is odd
  if ones%2 == 0 {
    b |= 0x80
  } else {
    b &^= 0x80
  }
  return b
}

type DESSubKeys struct {
  Keys [16]uint64
}

func NewDESSubKeys(keys [16]uint64) *DESSubKeys {
  return &DESSubKeys{
    Keys: keys,
  }
}

var leftPermutation = []uint8{
  57, 49, 41, 33, 25, 17, 9, 1,
  58, 50, 42, 34, 26, 18, 10, 2,
  59, 51, 43, 35, 27, 19, 11, 3,
  60, 52, 44, 36,
}

var rightPermutation = []uint8{
  63, 55, 47, 39, 31, 23, 15, 7,
  62, 54, 46, 38, 30, 22, 14, 6,
  61, 53, 45, 37, 29, 21, 13, 5,
```

```
  28, 20, 12, 4,
}

var keyResultPermutation = []uint8{
  14, 17, 11, 24, 1, 5,
  3, 28, 15, 6, 21, 10,
  23, 19, 12, 4, 26, 8,
  16, 7, 27, 20, 13, 2,
  41, 52, 31, 37, 47, 55,
  30, 40, 51, 45, 33, 48,
  44, 49, 39, 56, 34, 53,
  46, 42, 50, 36, 29, 32,
}

var shifts = []uint8{1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1}

func (de *DESKey) GenerateSubKeys() *DESSubKeys {
  var subKeys [16]uint64

  var c, d uint64
  c = perm.PermuteBlock(de.Key, leftPermutation)
  d = perm.PermuteBlock(de.Key, rightPermutation)

  for i := 0; i < 16; i++ {
    c = perm.RotateLeftBase(c, int(shifts[i]), 28)
    d = perm.RotateLeftBase(d, int(shifts[i]), 28)
    res := (c << 28) | d
    subKeys[i] = perm.PermuteBlock(res, keyResultPermutation)
  }
  return NewDESSubKeys(subKeys)
}

func (de *DESKey) Dump(writer io.Writer) error {
  _, err := fmt.Fprintf(writer, "DES Key:\n")
  if err != nil {
    return err
  }
  _, err = fmt.Fprintf(writer, "%d\n", de.Key)
  if err != nil {
    return err
```

```
  }

  return nil
}

func LoadDESKey(reader io.Reader) (*DESKey, error) {
  var key uint64
  _, err := fmt.Fscanf(reader, "DES Key:\n")
  if err != nil {
    return nil, err
  }
  _, err = fmt.Fscanf(reader, "%d\n", &key)
  if err != nil {
    return nil, err
  }

  return NewDESKey(key), nil
}
```

Листинг 2.2 — Код константных таблиц

```
package des

const BlockSize = 8

var sTable = [8][4][16]uint8{
  // S1
  {
    {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},
  },
  // S2
  {
    {15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10},
    {3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5},
    {0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15},
    {13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9},
  },
  // S3
```

```
{
  {10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8},
  {13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1},
  {13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7},
  {1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12},
},
// S4
{
  {7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15},
  {13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9},
  {10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4},
  {3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14},
},
// S5
{
  {2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9},
  {14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6},
  {4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14},
  {11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3},
},
// S6
{
  {12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11},
  {10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8},
  {9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6},
  {4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13},
},
// S7
{
  {4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1},
  {13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6},
  {1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2},
  {6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12},
},
// S8
{
  {13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7},
  {1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2},
  {7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8},
  {2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11},
```

```
  },
}
```

Листинг 2.3 — Код шифрования и расшифровки блока

```
package des

import "info-security/internal/utils/perm"

var initPermutation = []uint8{
  58, 50, 42, 34, 26, 18, 10, 2,
  60, 52, 44, 36, 28, 20, 12, 4,
  62, 54, 46, 38, 30, 22, 14, 6,
  64, 56, 48, 40, 32, 24, 16, 8,
  57, 49, 41, 33, 25, 17, 9, 1,
  59, 51, 43, 35, 27, 19, 11, 3,
  61, 53, 45, 37, 29, 21, 13, 5,
  63, 55, 47, 39, 31, 23, 15, 7,
}

var finalPermutation = []uint8{
  40, 8, 48, 16, 56, 24, 64, 32,
  39, 7, 47, 15, 55, 23, 63, 31,
  38, 6, 46, 14, 54, 22, 62, 30,
  37, 5, 45, 13, 53, 21, 61, 29,
  36, 4, 44, 12, 52, 20, 60, 28,
  35, 3, 43, 11, 51, 19, 59, 27,
  34, 2, 42, 10, 50, 18, 58, 26,
  33, 1, 41, 9, 49, 17, 57, 25,
}

func (k *DESSubKeys) encryptBlock(block uint64) uint64 {
  var res uint64

  initBlock := perm.PermuteBlock(block, initPermutation)
  li := uint32(initBlock >> 32)
  ri := uint32(initBlock & 0x00000000FFFFFFFF)

  for i := 0; i < 16; i++ {
    nextLi := ri
    ri = li ^ feistel(ri, k.Keys[i])
```

```
    li = nextLi
  }

  res = uint64(ri)<<32 | uint64(li)

  return perm.PermuteBlock(res, finalPermutation)
}

func (k *DESSubKeys) decryptBlock(block uint64) uint64 {
  var res uint64

  initBlock := perm.PermuteBlock(block, initPermutation)
  li := uint32(initBlock >> 32)
  ri := uint32(initBlock & 0x00000000FFFFFFFF)

  for i := 0; i < 16; i++ {
    nextLi := ri
    ri = li ^ feistel(ri, k.Keys[15-i])
    li = nextLi
  }
  res = uint64(ri)<<32 | uint64(li)
  return perm.PermuteBlock(res, finalPermutation)
}

var extendPermutation = []uint8{
  32, 1, 2, 3, 4, 5,
  4, 5, 6, 7, 8, 9,
  8, 9, 10, 11, 12, 13,
  12, 13, 14, 15, 16, 17,
  16, 17, 18, 19, 20, 21,
  20, 21, 22, 23, 24, 25,
  24, 25, 26, 27, 28, 29,
  28, 29, 30, 31, 32, 1,
}

var pPermutation = []uint8{
  16, 7, 20, 21,
  29, 12, 28, 17,
  1, 15, 23, 26,
  5, 18, 31, 10,
```

```
  2, 8, 24, 14,
  32, 27, 3, 9,
  19, 13, 30, 6,
  22, 11, 4, 25,
}

func feistel(ri uint32, ki uint64) uint32 {
  var exri uint64 = perm.PermuteBlock(uint64(ri), extendPermutation)
  var xored uint64 = exri ^ ki
  var sRes uint32
  for i := 0; i < 8; i++ {
    bits := (xored >> (42 - i*6)) & 0x3F
    row := ((bits & 0b100000) >> 4) | (bits & 1)
    col := (bits & 0b011110) >> 1
    sRes = (sRes << 4) | uint32(sTable[i][row][col])
  }

  return uint32(perm.PermuteBlock(uint64(sRes), pPermutation))
}
```

Листинг 2.4 — Код шифрования и расшифровки текста

```
package des

import (
  "encoding/binary"
  "errors"
  "io"
)

var ErrWrongEncryptedSize = errors.New("wrong encrypted size")

func (k *DESKey) Encrypt(reader io.Reader, writer io.Writer) error {
  subKeys := k.GenerateSubKeys()

  plain, err := io.ReadAll(reader)
  if err != nil {
    return err
  }

  // padding
```

```
  padding := BlockSize - (len(plain) % BlockSize)
  if padding == 0 {
    padding = BlockSize
  }
  for i := 0; i < padding; i++ {
    plain = append(plain, byte(padding))
  }

  // per-block encryption
  for i := 0; i < len(plain); i += BlockSize {
    block := binary.BigEndian.Uint64(plain[i : i+BlockSize])
    encrypted := subKeys.encryptBlock(block)
    var out [BlockSize]byte
    binary.BigEndian.PutUint64(out[:], encrypted)
    if _, err := writer.Write(out[:]); err != nil {
      return err
    }
  }
  return nil
}

func (k *DESKey) Decrypt(reader io.Reader, writer io.Writer) error {
  subKeys := k.GenerateSubKeys()

  cipher, err := io.ReadAll(reader)
  if err != nil {
    return err
  }
  if len(cipher)%BlockSize != 0 {
    return ErrWrongEncryptedSize
  }

  var plain []byte
  for i := 0; i < len(cipher); i += BlockSize {
    block := binary.BigEndian.Uint64(cipher[i : i+BlockSize])
    decrypted := subKeys.decryptBlock(block)
    var out [BlockSize]byte
    binary.BigEndian.PutUint64(out[:], decrypted)
    plain = append(plain, out[:]...)
  }
```

```
  // padding
  if len(plain) == 0 {
    return errors.New("empty plaintext")
  }
  padding := int(plain[len(plain)-1])
  if padding <= 0 || padding > BlockSize || padding > len(plain) {
    return errors.New("invalid padding")
  }
  // per-block decryption
  for i := 0; i < padding; i++ {
    if plain[len(plain)-1-i] != byte(padding) {
      return errors.New("invalid padding")
    }
  }
  plain = plain[:len(plain)-padding]

  _, err = writer.Write(plain)
  return err
}
```

2.3 Пример работы программы

Разработанное ПО представляет из себя 2 программы с интерфейсом командной строки.

Первая программа создаёт случайный ключ. Она принимает один аргумент:

— **-out** – файл, в который нужно сохранить ключ.

```
go run ./cmd/des/gen --help
Usage of /home/impervguin/.cache/go-build/7c/7c7f9d0ba79fae1ad5d07fadc3c496ccdd7e171d8
  -out string
        key file name. default: ./config/des/key.txt (default "./config/des/key.txt")
```

Рисунок 2.1 — Описание аргументов программы генерации ключа

Вторая программа использует ключ для шифровки или расшифровки произвольного файла. Она принимает 4 аргумента:

— **-in** – имя файла с открытым текстом;

— **-out** – имя файла, куда сохранить зашифрованный текст;

— **-key** – имя файла с ключом.

— **-op** – тип операции. Возможные значения encrypt – шифрование, decrypt – расшифровка.

```
go run ./cmd/des/crypt --help
Usage of /home/impervguin/.cache/go-build/f9/f9295b972522b1f361f458663137c85d6ff71908a
  -in string
        input file name
  -key string
        key file name. default: ./config/des/key.txt (default "./config/des/key.txt")
  -op string
        operation type. default: encrypt. possible values: encrypt, decrypt (default "
  -out string
        output file name
```

Рисунок 2.2 — Описание аргументов программы шифрования и расшифровки

В качестве примера программы был взят текст из книги «Принципы юнит-тестирования» автора Владимир Хориков, длиной примерно 13000 символов.

На рисунках 2.3- 2.4 представлен процесс шифровки и расшифровки указанного текста.

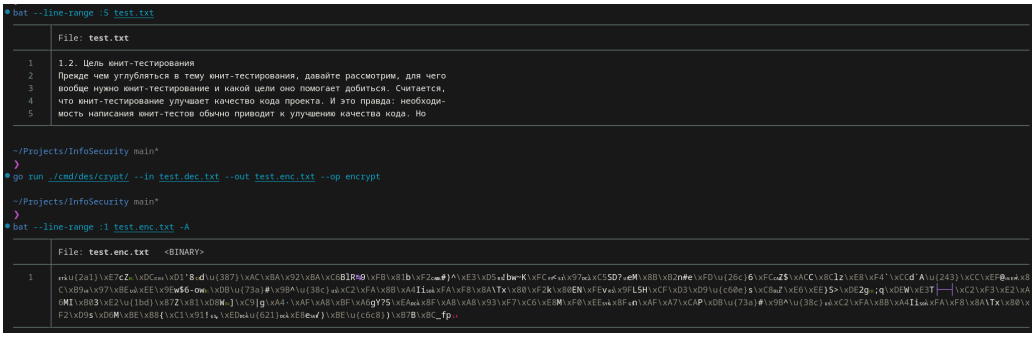

Рисунок 2.3 — Шифрование текстового файла

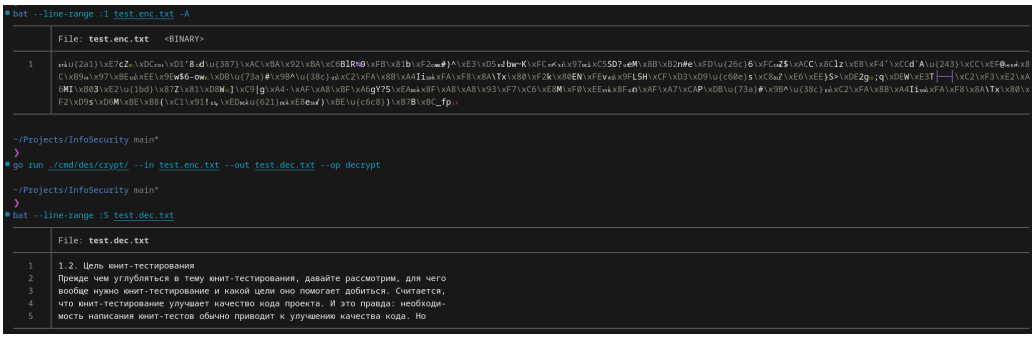

Рисунок 2.4 — Расшифровка текстового файла

ЗАКЛЮЧЕНИЕ

В результате лабораторной работы была разработана программная реализация алгоритма DES.

Были выполнены следующие задачи:

1) проанализирован принцип работы алгоритма DES;
2) реализованы описанные алгоритмы шифрования и расшифровки;
3) проведено шифрование и расшифровка текстового файла;

Все поставленные цели и задачи были выполнены.