Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное 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 № 6
по дисциплине «Защита информации»

Тема Реализация алгоритма симметричного шифрования (DES)

Студент Пермякова Е. Д.

Группа ИУ7-72Б

Преподаватели Руденкова Ю. С.

Москва, 2025

ВВЕДЕНИЕ

Целью данной работы является разработка алгоритма симметричного шифрования (DES). Шифрование и расшифровка архивных файлов.

Для достижения поставленной цели требуется решить следующие задачи:

1) описать алгоритм симметричного шифрования (DES);

2) реализовать виде программы алгоритм симметричного шифрования (DES);

1 Теоретическая часть

Виды симметричного шифрования:

1) Поточные: шифруют данные побитово/побайтово (RC4);

2) Блочные: шифруют данные блоками фиксированного размера (DES, AES);

Алгоритмы перестановки — это методы, которые изменяют порядок следования элементов (но не их значения) посредством присваивания и перестановки их значений. Пример: IP в DES.

Алгоритмы подстановки — это методы шифрования, в которых элементы исходного открытого текста заменяются зашифрованным текстом в соответствии с некоторым правилом. Пример: шифр Цезаря.

Алгоритм DES использует методы перестановки и подстановки.

2 Описание алгоритма симметричного шифрования (DES)

На рисунке 2.1 приведена схема генерации подключей, которая выполняется перед началом шифрования по алгоритму DES.

Рисунок 2.1 – Схема генерации генерации подключей

На рисунке 2.2 приведена схема алгоритма симметричного шифрования (DES).

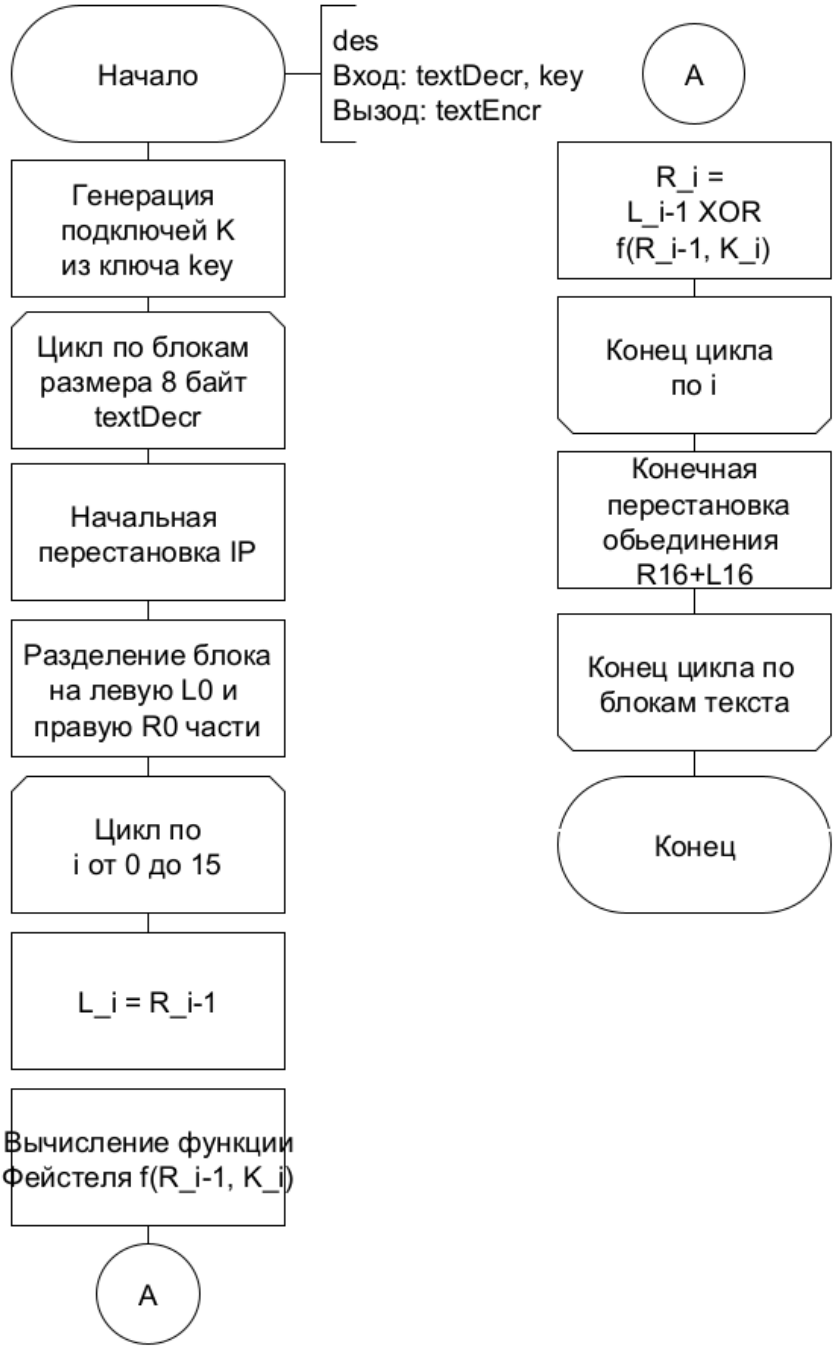

Рисунок 2.2 – Схема алгоритма симметричного шифрования (DES)

3 Пример работы алгоритма симметричного шифрования (DES)

Для архивирования и разархивирования файлов использовалась программа WinRAR [1].

На рисунках 3.1-3.4 приведен пример работы алгоритма симметричного шифрования (DES) для шифрования и расшифровки архивных файлов.

```
kathrine@Viva:~/vuz/InfoSec/is_5/src$ go run ./des.go
Использование:
  Шифрование: go run des.go encrypt <файл_ключа> <входной_файл> <выходной_файл>
  Дешифрование: go run des.go decrypt <файл_ключа> <входной_файл> <выходной_файл>
  Генерация ключа: go run des.go genkey <файл_ключа>

kathrine@Viva:~/vuz/InfoSec/is_5/src$ go run des.go genkey ./key.txt
Ключ успешно сгенерирован и сохранен в: ./key.txt
Hex-представление: A4AAC0663D5D7249
kathrine@Viva:~/vuz/InfoSec/is_5/src$ cat ./key.txt
a4aac0663d5d7249kathrine@Viva:~/vuz/InfoSec/is_5/src$
kathrine@Viva:~/vuz/InfoSec/is_5/src$ go run des.go encrypt ./key.txt ../data/text.rar ../data/encrypted.rar
Загружен ключ: A4AAC0663D5D7249
Файл успешно зашифрован: ../data/text.rar -> ../data/encrypted.rar
kathrine@Viva:~/vuz/InfoSec/is_5/src$ go run des.go decrypt ./key.txt ../data/encrypted.rar ../data/decrypted.rar
Загружен ключ: A4AAC0663D5D7249
Файл успешно расшифрован: ../data/encrypted.rar -> ../data/decrypted.rar
```

Рисунок 3.1 – Пример работы алгоритма симметричного шифрования (DES)

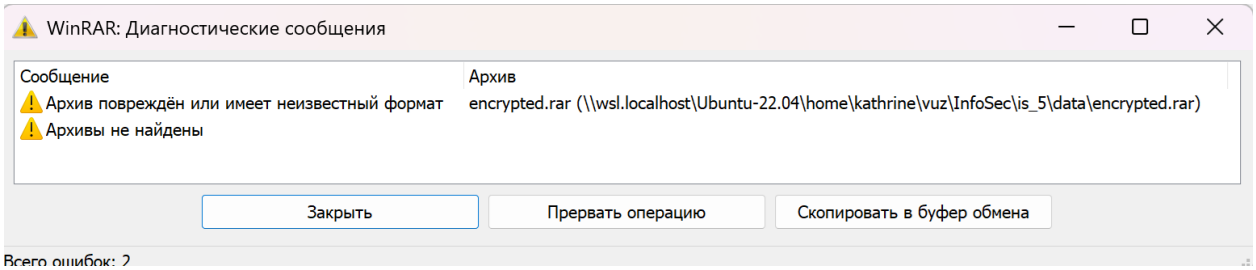

Рисунок 3.2 – Пример работы алгоритма симметричного шифрования (DES) – попытка разархивировать зашифрованный архив

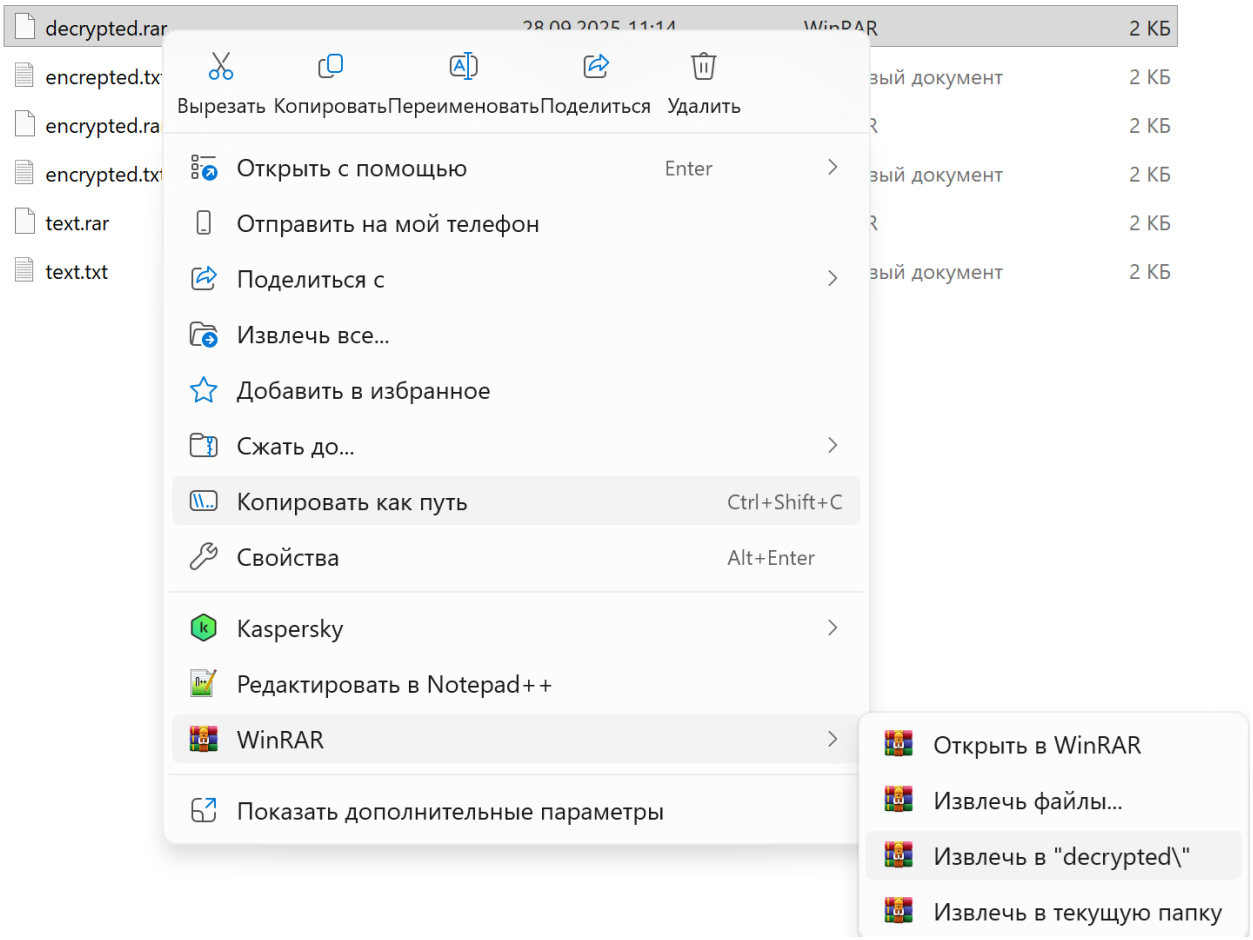

Рисунок 3.3 – Пример работы алгоритма симметричного шифрования (DES) – разархивирование расшифрованного архива

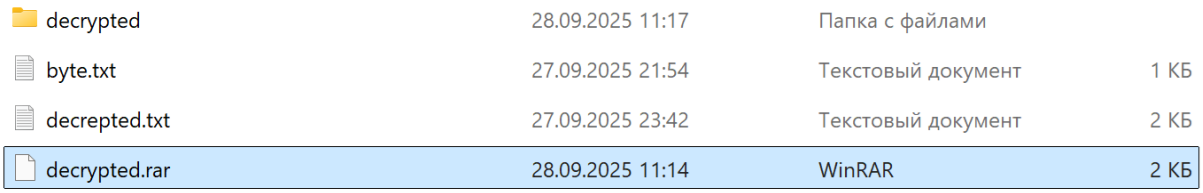

Рисунок 3.4 – Пример работы алгоритма симметричного шифрования (DES) – расшифрованный архив был успешно разархивирован

4 Реализация алгоритма симметричного шифрования (DES)

В качестве средства реализации алгоритма симметричного шифрования (DES) был выбран язык Go.

```
type DES struct {
    subkeys [16]uint64
}

func NewDES(key uint64) *DES {
    des := &DES{}
    des.generateSubkeys(key)
    return des
}

func initialPermutation(block uint64) uint64 {
    ipTable := [64]int{
        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,
    }
    return permute(block, ipTable[:], 64)
}

func finalPermutation(block uint64) uint64 {
    fpTable := [64]int{
        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,
    }
    return permute(block, fpTable[:], 64)
}
```

```
func expansionPermutation(right uint32) uint64 {
    eTable := [48]int{
        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,
    }
    return uint64(permute32(uint64(right), eTable[:], 32))
}

func pPermutation(data uint32) uint32 {
    pTable := [32]int{
        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,
    }
    return uint32(permute32(uint64(data), pTable[:], 32))
}

func sBoxSubstitution(data uint64) uint32 {
    sBoxes := [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},
```

```
    },
    }

    var result uint32
    for i := 0; i < 8; i++ {
        chunk := (data >> (42 - 6*i)) & 0x3F
        row := ((chunk & 0x20) >> 4) | (chunk & 0x01)
        col := (chunk >> 1) & 0x0F
        sVal := sBoxes[i][row][col]
        result = (result << 4) | uint32(sVal)
    }

    return result
}

func feistelFunction(right uint32, subkey uint64) uint32 {
    expanded := expansionPermutation(right)
    xored := expanded ^ subkey
    substituted := sBoxSubstitution(xored)
    return pPermutation(substituted)
}

func (des *DES) generateSubkeys(key uint64) {
    pc1Table := [56]int{
        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, 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,
    }
    pc2Table := [48]int{
        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,
    }
    shifts := [16]int{1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2,
```

```
        2, 1}

    permutedKey := permute(key, pc1Table[:], 64)

    c := (permutedKey >> 28) & 0x0FFFFFFF
    d := permutedKey & 0x0FFFFFFF

    for i := 0; i < 16; i++ {
        c = ((c << shifts[i]) | (c >> (28 - shifts[i]))) & 0
           x0FFFFFFF
        d = ((d << shifts[i]) | (d >> (28 - shifts[i]))) & 0
           x0FFFFFFF

        combined := (c << 28) | d
        des.subkeys[i] = permute(combined, pc2Table[:], 56)
    }
}

func (d *DES) encryptBlock(block uint64) uint64 {
    block = initialPermutation(block)

    left := uint32(block >> 32)
    right := uint32(block & 0xFFFFFFFF)

    for i := 0; i < 16; i++ {
        nextLeft := right
        fResult := feistelFunction(right, d.subkeys[i])
        nextRight := left ^ fResult

        left = nextLeft
        right = nextRight
    }

    combined := (uint64(right) << 32) | uint64(left)

    return finalPermutation(combined)
}

func (d *DES) decryptBlock(block uint64) uint64 {
    temp := d.subkeys
    for i := 0; i < 8; i++ {
```

```
        d.subkeys[i], d.subkeys[15-i] = d.subkeys[15-i], d.
            subkeys[i]
    }
    result := d.encryptBlock(block)
    d.subkeys = temp

    return result
}

func permute(data uint64, table []int, inputSize int) uint64 {
    var result uint64
    for i, pos := range table {
        bit := (data >> (inputSize - pos)) & 1
        result |= bit << (uint(len(table)) - 1 - uint(i))
    }
    return result
}

func permute32(data uint64, table []int, inputSize int) uint64
   {
    var result uint64
    for i, pos := range table {
        bit := (data >> (inputSize - pos)) & 1
        result |= bit << (uint(len(table)) - 1 - uint(i))
    }
    return result
}

func (d *DES) encryptFile(inputPath, outputPath string) error {
    inputFile, err := os.Open(inputPath)
    if err != nil {
        return err
    }
    defer inputFile.Close()

    outputFile, err := os.Create(outputPath)
    if err != nil {
        return err
    }
    defer outputFile.Close()
```

```
    buffer := make([]byte, 8)
    for {
        n, err := inputFile.Read(buffer)
        if err != nil && err != io.EOF {
            return err
        }
        if n == 0 {
            break
        }
        if n < 8 {
            for i := n; i < 8; i++ {
                buffer[i] = 0
            }
        }
        block := binary.BigEndian.Uint64(buffer)
        encryptedBlock := d.encryptBlock(block)
        encryptedBytes := make([]byte, 8)
        binary.BigEndian.PutUint64(encryptedBytes,
           encryptedBlock)
        _, err = outputFile.Write(encryptedBytes)
        if err != nil {
            return err
        }
        if err == io.EOF {
            break
        }
    }
    return nil
}


func (d *DES) decryptFile(inputPath, outputPath string) error {
    inputFile, err := os.Open(inputPath)
    if err != nil {
        return err
    }
    defer inputFile.Close()
    outputFile, err := os.Create(outputPath)
    if err != nil {
        return err
    }
```

```
    defer outputFile.Close()
    buffer := make([]byte, 8)
    for {
        n, err := inputFile.Read(buffer)
        if err != nil && err != io.EOF {
            return err
        }
        if n == 0 {
            break
        }
        block := binary.BigEndian.Uint64(buffer)
        decryptedBlock := d.decryptBlock(block)
        decryptedBytes := make([]byte, 8)
        binary.BigEndian.PutUint64(decryptedBytes,
           decryptedBlock)

        if err == io.EOF || n < 8 {
            decryptedBytes = decryptedBytes[:n]
        }
        _, err = outputFile.Write(decryptedBytes)
        if err != nil {
            return err
        }
        if err == io.EOF {
            break
        }
    }
    return nil
}

func readKeyFromFile(keyFilePath string) (uint64, error) {
    keyData, err := os.ReadFile(keyFilePath)
    if err != nil {
        return 0, fmt.Errorf("error: %v", err)
    }
    keyStr := strings.TrimSpace(string(keyData))
    keyStr = strings.ReplaceAll(keyStr, " ", "")
    keyStr = strings.ReplaceAll(keyStr, "\n", "")
    keyStr = strings.ReplaceAll(keyStr, "\r", "")
    keyStr = strings.ReplaceAll(keyStr, "0x", "")
    keyStr = strings.ReplaceAll(keyStr, "0X", "")
```

```
    if len(keyStr) != 16 {
        return 0, fmt.Errorf("error key len %d", len(keyStr))
    }
    keyBytes, err := hex.DecodeString(keyStr)
    if err != nil {
        return 0, fmt.Errorf("DecodeString: %v", err)
    }

    if len(keyBytes) != 8 {
        return 0, fmt.Errorf("len(keyBytes) != 8")
    }
    return binary.BigEndian.Uint64(keyBytes), nil
}

func generateAndSaveKey(keyFilePath string) (uint64, error) {
    keyBytes := make([]byte, 8)
    _, err := rand.Read(keyBytes)
    if err != nil {
        return 0, fmt.Errorf("err: %v", err)
    }
    key := binary.BigEndian.Uint64(keyBytes)
    keyHex := hex.EncodeToString(keyBytes)
    err = os.WriteFile(keyFilePath, []byte(keyHex), 0600)
    if err != nil {
        return 0, fmt.Errorf("err: %v", err)
    }
    return key, nil
}
```

Листинг 4.1 – Реализация алгоритма шифрования и расшифровки с открытым ключом файла

ЗАКЛЮЧЕНИЕ

В ходе лабораторной работы был реализован алгоритм симметричного шифрования (DES).

В процессе выполнения данной работы были выполнены все задачи:

1) описать алгоритм симметричного шифрования (DES);

2) реализовать виде программы алгоритм симметричного шифрования (DES);

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. WinRAR. URL: Режим доступа: https://www.win-rar.com/start.html?&L=4 (дата обращения: 14.09.2025).