



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 № 7
по дисциплине «Защита информации»**

Тема Реализация алгоритма симметричного шифрования (AES).

Студент Шахнович Дмитрий Сергеевич

Группа ИУ7-72Б

Преподаватель Руденкова Ю.С.

Москва, 2025

Содержание

| | |
|--|----------|
| ВВЕДЕНИЕ | 3 |
| 1 Аналитическая часть | 4 |
| 1.1 Определения | 4 |
| 1.2 Алгоритм AES | 6 |
| 1.2.1 Основные определения AES | 6 |
| 1.2.2 Генерация раундовых ключей | 7 |
| 1.2.3 Алгоритм шифрования блока | 7 |
| 1.2.4 Алгоритм расшифровки блока | 8 |
| 2 Технологическая часть | 9 |
| 2.1 Средства реализации | 9 |
| 2.2 Код разработанной программы | 9 |
| 2.3 Пример работы программы | 19 |

ВВЕДЕНИЕ

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

Задачи:

- 1) проанализировать алгоритмы шифрования и расшифровки AES;
- 2) реализовать программу шифрования симметричным алгоритмом AES;
- 3) обеспечить шифрование и расшифровку произвольного файла и тар-
архива 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.

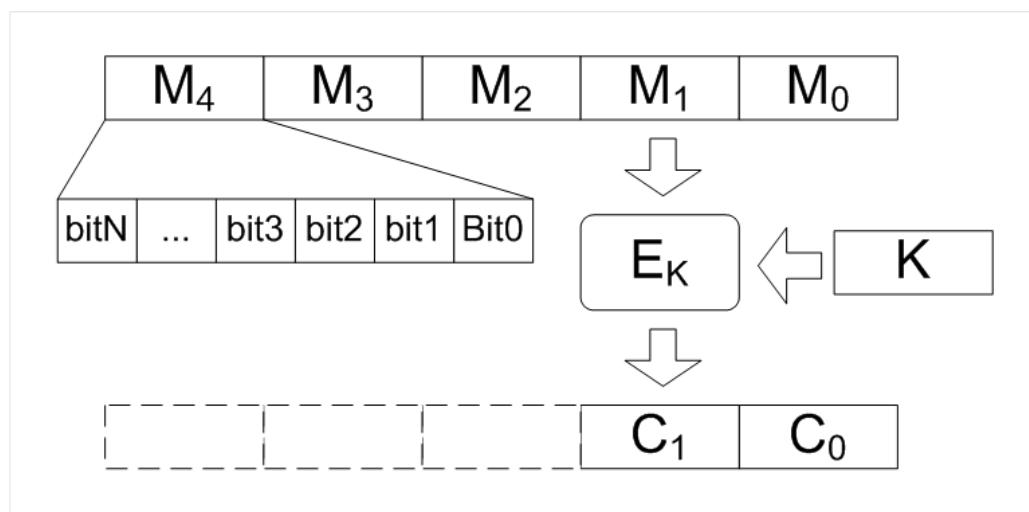


Рисунок 1.1 — Обобщённая схема блочного шифра

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.

Классификация автоматизированных систем, с точки зрения информационной безопасности:

- 1) Класс 1 (AC-1) — Системы высшего класса защищённости – АС, предназначенные для обработки информации особой важности (в рамках конфиденциальной) или конфиденциальной информации, нарушение конфиденциальности которой может нанести катастрофический ущерб.
 - Использование средств защиты информации, прошедших обязательную сертификацию.
 - Межсетевые экраны (файрволы) не ниже 6-го класса.
 - Средства защиты от вредоносного кода не ниже 2-го класса.
 - Строжайший контроль учётных записей, парольная политика высокой сложности.
 - Протоколирование всех значимых событий безопасности.
 - Обязательное управление обновлениями и конфигурациями.
 - Защита от утечек по техническим каналам (электромагнитное излучение и т.д.).
- 2) Класс 2 (AC-2) — Системы среднего класса защищённости – АС, предназначенные для обработки конфиденциальной информации (например, персональные данные в большом объёме) или информации, нарушение целостности/доступности которой нанесёт значительный ущерб.
 - Использование сертифицированных СЗИ.
 - Межсетевые экраны не ниже 5-го класса.
 - Средства защиты от вредоносного кода не ниже 2-го класса.
 - Жёсткий контроль учётных записей и прав доступа.
 - Обязательное протоколирование.
 - Требования к резервному копированию и восстановлению.
- 3) Класс 3 (AC-3) — Системы базового класса защищённости – АС, пред-

назначенные для обработки общедоступной информации, нарушение безопасности которой (главным образом, целостности и доступности) нанесёт незначительный ущерб. Сюда же часто относят системы, обрабатывающие персональные данные, но не в большом объёме и не относящиеся к специальным категориям.

- Наличие межсетевого экрана.
- Наличие антивирусной защиты.
- Учёт пользователей и разграничение прав доступа.
- Регулярное обновление ПО.
- Обеспечение резервного копирования.

1.2 Алгоритм AES

AES – симметричный алгоритм блочного шифрования (размер блока 128 бит, ключ 128/192/256 бит), принятый в качестве стандарта шифрования правительством США по результатам конкурса AES. AES является стандартом, основанным на алгоритме Rijndael. Для AES длина input (блока входных данных) и State (переходного состояния блока) постоянна и равна 128 бит, а длина шифролюча K составляет 128, 192, или 256 бит.

1.2.1 Основные определения AES

- BlockSize = 16. Блок данных — 16 байт (128 бит). Вся блочная обработка — по 16-байтовым блокам.
- Nb = 4 Число колонок (32 битных слов) состояния.
- KeySize — длина исходного ключа в байтах. Допустимые значения 16, 24 и 32.
- Nk = KeySize / 4 — число 32-бит слов в ключе.
- Nr = Nk + 6 — число раундов шифрования в алгоритме.
- State Состояние блока — массив из 16 байт, интерпретируемый как матрица 4×4 байта. Промежуточное состояние шифруемого блока.

Также в алгоритме используется несколько констант:

- sBox — таблица подстановок;
- invSBox — обратная таблица подстановок;
- rcon — массив констант, для генерации раундовых ключей.

1.2.2 Генерация раундовых ключей

Каждый раундовый ключ представляет из себя массив байт размером 16 байт, то есть эквивалентен в размере блоку шифрования, а всего этих ключей. Алгоритм генерации ключей имеет вид:

- 1) Первый раундовый ключ равен первым 16-ти байтам ключа шифрования;
- 2) Цикл по $i = 1$ до Nr :
 - 2.1) $temp$ – массив из 4 байт, равный последним 4 байтам ключа $i-1$;
 - 2.2) циклически сдвинуть байты $temp$ влево на 1;
 - 2.3) по специальной таблице переставить биты $temp$;
 - 2.4) XOR первого байта $temp$ с i -го константой $rcon$;
 - 2.5) $currentKey$ – массив из 16 байт, i -й раундовый ключ;
 - 2.6) $currentKey[0..3] = prevKey[0..3] \text{ XOR } temp[0..3]$;
 - 2.7) для $j = 4..15$: $currentKey[j] = currentKey[j-4] \text{ XOR } prevKey[j]$

1.2.3 Алгоритм шифрования блока

Размер блока для шифрования алгоритмом AES – 128 бит

Алгоритм шифрования отдельного блока имеет следующий вид:

- 1) Побитовый xor блока с первым раундовым ключом
- 2) Цикл по i от 1 до $Nr - 1$:
 - 2.1) замена всех байтов блока в соответствии с таблицей замены ($subBytes$);
 - 2.2) представить блок как матрица 4×4 состоящую из матриц блоков, и провести циклическое смещение каждой строки матрицы на число, равное номеру этой строки ($shiftRows$);
 - 2.3) представить каждый столбец матрицы как полином третьей степени в поле Галуа и перемножить его с заданным многочленом из того же поля ($mixColumns$);
 - 2.4) побитовый xor с i -м раундовым ключом ($addRoundKey$);
- 3) замена всех байтов блока в соответствии с таблицей замены ($subBytes$);
- 4) представить блок как матрица 4×4 состоящую из матриц блоков, и провести циклическое смещение каждой строки матрицы на число, равное

номеру этой строки (shiftRows);

5) побитовый xor с последним раундовым ключом (addRoundKey);

Результатом является блок размером 128 бита – зашифрованный блок.

1.2.4 Алгоритм расшифровки блока

Алгоритм расшифровки по структуре совпадает с алгоритмом шифрования, но имеет ряд отличий:

- вместо subBytes, shiftRows, mixColumns используют обратные к ним операции;
- раундовые ключи используются в обратном порядке;

Результатом является блок размером 128 бит – блок открытого текста.

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- 2.2 представлена разработанная программа.

Листинг 2.1 — Код генерации ключей

```
package aes

import (
    "crypto/rand"
    "fmt"
    "io"
    "slices"
)

type AESKey struct {
    key        []byte
    roundKeys [][]byte
}

var (
    keyBitSizes = []int{128, 192, 256}
    keySizes    = []int{16, 24, 32}
)

func NewAESKey(key []byte) (*AESKey, error) {
    if !slices.Contains(keySizes, len(key)) {
        return nil, fmt.Errorf("invalid key size")
    }
    a := &AESKey{key: key}
    a.roundKeys = a.keyExpansion()
    return a, nil
}

func (a *AESKey) KeySize() int {
```

```

    return len(a.key)
}

func GenerateAESKey(bitsize int) (*AESKey, error) {
    if !slices.Contains(keyBitSizes, bitsize) {
        return nil, fmt.Errorf("invalid key size")
    }
    key := make([]byte, bitsize/8)
    _, err := rand.Read(key)
    if err != nil {
        return nil, err
    }
    return NewAESKey(key)
}

func (a *AESKey) Dump(writer io.Writer) error {
    _, err := writer.Write(a.key)
    return err
}

func LoadAESKey(reader io.Reader) (*AESKey, error) {
    key, err := io.ReadAll(reader)
    if err != nil {
        return nil, err
    }
    return NewAESKey(key)
}

```

Листинг 2.2 — Код генерации раундовых ключей

```

package aes

func (a *AESKey) keyExpansion() [][]byte {
    nk := a.KeySize() / 4
    nr := nk + 6

    roundKeys := make([][]byte, nr+1)
    for i := range roundKeys {
        roundKeys[i] = make([]byte, BlockSize)
    }
}

```

```

copy( roundKeys[0] , a.key )

for i := 1; i <= nr; i++ {
    prevKey := roundKeys[i-1]
    currentKey := roundKeys[i]

    temp := make([]byte, 4)
    copy(temp, prevKey[BlockSize-4:BlockSize])
    temp = subWord(rotWord(temp))
    temp[0] ^= byte(rcon[i] >> 24)

    for j := 0; j < 4; j++ {
        currentKey[j] = prevKey[j] ^ temp[j]
    }

    for j := 4; j < BlockSize; j++ {
        currentKey[j] = currentKey[j-4] ^ prevKey[j]
    }
}

return roundKeys
}

func rotWord(word []byte) []byte {
    return []byte{word[1], word[2], word[3], word[0]}
}

func subWord(word []byte) []byte {
    result := make([]byte, 4)
    for i := 0; i < 4; i++ {
        result[i] = sBox[word[i]]
    }
    return result
}

```

Листинг 2.3 — Код константных таблиц

```

package aes

const (
    BlockSize = 16
    Nb        = 4

```

```

)

var sBox = [256]byte{
    0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67,
    0x2b, 0xfe, 0xd7, 0xab, 0x76,
    0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2,
    0xaf, 0x9c, 0xa4, 0x72, 0xc0,
    0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5,
    0xf1, 0x71, 0xd8, 0x31, 0x15,
    0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80,
    0xe2, 0xeb, 0x27, 0xb2, 0x75,
    0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6,
    0xb3, 0x29, 0xe3, 0x2f, 0x84,
    0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe,
    0x39, 0x4a, 0x4c, 0x58, 0xcf,
    0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02,
    0x7f, 0x50, 0x3c, 0x9f, 0xa8,
    0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda,
    0x21, 0x10, 0xff, 0xf3, 0xd2,
    0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e,
    0x3d, 0x64, 0x5d, 0x19, 0x73,
    0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8,
    0x14, 0xde, 0x5e, 0x0b, 0xdb,
    0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac,
    0x62, 0x91, 0x95, 0xe4, 0x79,
    0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4,
    0xea, 0x65, 0x7a, 0xae, 0x08,
    0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74,
    0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
    0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57,
    0xb9, 0x86, 0xc1, 0x1d, 0x9e,
    0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87,
    0xe9, 0xce, 0x55, 0x28, 0xdf,
    0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d,
    0x0f, 0xb0, 0x54, 0xbb, 0x16,
}

var invSBox = [256]byte{
    0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3,
    0x9e, 0x81, 0xf3, 0xd7, 0xfb,
}

```

```

0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43,
    0x44, 0xc4, 0xde, 0xe9, 0xcb,
0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95,
    0x0b, 0x42, 0xfa, 0xc3, 0x4e,
0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2,
    0x49, 0x6d, 0x8b, 0xd1, 0x25,
0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c,
    0xcc, 0x5d, 0x65, 0xb6, 0x92,
0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46,
    0x57, 0xa7, 0x8d, 0x9d, 0x84,
0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0xa, 0xf7, 0xe4, 0x58,
    0x05, 0xb8, 0xb3, 0x45, 0x06,
0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd,
    0x03, 0x01, 0x13, 0x8a, 0x6b,
0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf,
    0xce, 0xf0, 0xb4, 0xe6, 0x73,
0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37,
    0xe8, 0x1c, 0x75, 0xdf, 0x6e,
0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62,
    0x0e, 0xaa, 0x18, 0xbe, 0x1b,
0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0,
    0xfe, 0x78, 0xcd, 0x5a, 0xf4,
0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10,
    0x59, 0x27, 0x80, 0xec, 0x5f,
0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a,
    0x9f, 0x93, 0xc9, 0x9c, 0xef,
0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb,
    0x3c, 0x83, 0x53, 0x99, 0x61,
0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14,
    0x63, 0x55, 0x21, 0x0c, 0x7d,
}

var rcon = [11]uint32{
    0x00000000, 0x01000000, 0x02000000, 0x04000000, 0x08000000,
    0x10000000, 0x20000000, 0x40000000, 0x80000000, 0x1b000000,
    0x36000000,
}

```

Листинг 2.4 — Код шифрования и расшифровки блока

```
package aes
```

```

import "fmt"

func subBytes(state []byte) {
    for i := 0; i < BlockSize; i++ {
        state[i] = sBox[state[i]]
    }
}

func invSubBytes(state []byte) {
    for i := 0; i < BlockSize; i++ {
        state[i] = invSBox[state[i]]
    }
}

func shiftRows(state []byte) {
    state[1], state[5], state[9], state[13] = state[5], state[9],
    state[13], state[1]

    state[2], state[6], state[10], state[14] = state[10], state[14],
    state[2], state[6]

    state[3], state[7], state[11], state[15] = state[15], state[3],
    state[7], state[11]
}

func invShiftRows(state []byte) {
    state[1], state[5], state[9], state[13] = state[13], state[1],
    state[5], state[9]
    state[2], state[6], state[10], state[14] = state[10], state[14],
    state[2], state[6]
    state[3], state[7], state[11], state[15] = state[7], state[11],
    state[15], state[3]
}

func galoisMultiply(a, b byte) byte {
    var p byte
    var hiBitSet byte

    for i := 0; i < 8; i++ {

```

```

    if (b & 1) == 1 {
        p ^= a
    }

    hiBitSet = a & 0x80
    a <<= 1
    if hiBitSet == 0x80 {
        a ^= 0x1b
    }
    b >>= 1
}

return p
}

func mixColumns(state []byte) {
    for i := 0; i < 4; i++ {
        col := state[i*4 : i*4+4]
        a := make([]byte, 4)
        b := make([]byte, 4)

        for j := 0; j < 4; j++ {
            a[j] = col[j]
            b[j] = col[j] & 0x80
            if b[j] == 0x80 {
                b[j] = (col[j] << 1) ^ 0x1b
            } else {
                b[j] = col[j] << 1
            }
        }

        col[0] = b[0] ^ a[3] ^ a[2] ^ b[1] ^ a[1]
        col[1] = b[1] ^ a[0] ^ a[3] ^ b[2] ^ a[2]
        col[2] = b[2] ^ a[1] ^ a[0] ^ b[3] ^ a[3]
        col[3] = b[3] ^ a[2] ^ a[1] ^ b[0] ^ a[0]
    }
}

func invMixColumns(state []byte) {
    for i := 0; i < 4; i++ {

```

```

    col := state[i*4 : i*4+4]

    a := make([]byte, 4)
    for j := 0; j < 4; j++ {
        a[j] = col[j]
    }

    col[0] = galoisMultiply(a[0], 0x0e) ^ galoisMultiply(a[1], 0x0b)
    ^
    galoisMultiply(a[2], 0x0d) ^ galoisMultiply(a[3], 0x09)
    col[1] = galoisMultiply(a[0], 0x09) ^ galoisMultiply(a[1], 0x0e)
    ^
    galoisMultiply(a[2], 0x0b) ^ galoisMultiply(a[3], 0x0d)
    col[2] = galoisMultiply(a[0], 0x0d) ^ galoisMultiply(a[1], 0x09)
    ^
    galoisMultiply(a[2], 0x0e) ^ galoisMultiply(a[3], 0x0b)
    col[3] = galoisMultiply(a[0], 0x0b) ^ galoisMultiply(a[1], 0x0d)
    ^
    galoisMultiply(a[2], 0x09) ^ galoisMultiply(a[3], 0x0e)
}

}

func addRoundKey(state, roundKey []byte) {
    for i := 0; i < BlockSize; i++ {
        state[i] ^= roundKey[i]
    }
}

func (a *AESKey) encryptBlock(block []byte) ([]byte, error) {
    if len(block) != BlockSize {
        return nil, fmt.Errorf("invalid block size")
    }
    state := make([]byte, BlockSize)
    copy(state, block)

    nr := len(a.roundKeys) - 1

    addRoundKey(state, a.roundKeys[0])
    for round := 1; round <= nr-1; round++ {
        subBytes(state)
    }
}

```

```

    shiftRows(state)
    mixColumns(state)
    addRoundKey(state, a.roundKeys[round])
}
subBytes(state)
shiftRows(state)
addRoundKey(state, a.roundKeys[nr])
return state, nil
}

func (a *AESKey) decryptBlock(block []byte) ([]byte, error) {
if len(block) != BlockSize {
    return nil, fmt.Errorf("invalid block size")
}
state := make([]byte, BlockSize)
copy(state, block)

nr := len(a.roundKeys) - 1

addRoundKey(state, a.roundKeys[nr])
for round := nr - 1; round > 0; round-- {
    invShiftRows(state)
    invSubBytes(state)
    addRoundKey(state, a.roundKeys[round])
    invMixColumns(state)
}
invShiftRows(state)
invSubBytes(state)
addRoundKey(state, a.roundKeys[0])
return state, nil
}

```

Листинг 2.5 — Код шифрования и расшифровки текста

```

package aes

import "io"

func PKCS7Padding(data []byte, blockSize int) []byte {
padding := blockSize - (len(data) % blockSize)
padText := make([]byte, padding)

```

```

    for i := range padText {
        padText[i] = byte(padding)
    }
    return append(data, padText...)
}

func PKCS7Unpadding(data []byte) []byte {
    if len(data) == 0 {
        return data
    }
    padding := int(data[len(data)-1])
    if padding > len(data) {
        return data
    }
    return data[:len(data)-padding]
}

func (a *AESKey) Encrypt(reader io.Reader, writer io.Writer) error {
    text, err := io.ReadAll(reader)
    if err != nil {
        return err
    }
    paddedText := PKCS7Padding(text, BlockSize)

    for i := 0; i < len(paddedText); i += BlockSize {
        block := paddedText[i : i+BlockSize]
        encryptedBlock, err := a.encryptBlock(block)
        if err != nil {
            return err
        }
        _, err = writer.Write(encryptedBlock)
        if err != nil {
            return err
        }
    }
    return nil
}

func (a *AESKey) Decrypt(reader io.Reader, writer io.Writer) error {
    text, err := io.ReadAll(reader)

```

```

if err != nil {
    return err
}
decryptedText := make([]byte, 0, len(text))
var i int
for i = 0; i < len(text); i += BlockSize {
    block := text[i : i+BlockSize]
    decryptedBlock, err := a.decryptBlock(block)
    if err != nil {
        return err
    }
    decryptedText = append(decryptedText, decryptedBlock...)
}
unpaddedText := PKCS7Unpadding(decryptedText)
_, err = writer.Write(unpaddedText)
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.
- **-bitsize** – размер ключа в битах.

```

$ go run ./cmd/aes/gen/ --help
Usage of /home/imperguin/.cache/go-build/5c/5cd82064f70210985a405f18bd024a5b28ff53f4f8ec2395b660f37b9e020ee4-d/gen:
  -bitsize int
      key size in bits. Supported sizes: 128, 192, 256 (default 128)
  -out string
      output file. Default: ./config/aes/key (default "./config/aes/key")

```

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/aes/crypt
Usage of /tmp/go-build3507085235/b001/exe/crypt:
  -in string
    input file name
  -key string
    key file name. default: ./config/aes/key (default "./config/aes/key")
  -op string
    operation type. default: encrypt. possible values: encrypt, decrypt (default "encrypt")
  -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.

```
* /Projects/InfoSecurity main* ↑
>
* bat --line-range :5 test.txt
File: test.txt
1 1.2. Цель юнит-тестирования
2 Прежде чем углубляться в тему юнит-тестирования, давайте рассмотрим, для чего
3 вообще нужно юнит-тестирование и какой цели оно помогает добиться. Считается,
4 что юнит-тестирование улучшает качество кода проекта. И это правда: необходимость
5 написания юнит-тестов обычно приводит к улучшению качества кода. Но

* /Projects/InfoSecurity main* ↑
>
* go run ./cmd/aes/crypt/main.go -in test.txt -out test.enc.txt -op encrypt
* /Projects/InfoSecurity main* ↑
>
* bat --line-range :1 test.enc.txt -A
File: test.enc.txt <BINARY>
1
```

Рисунок 2.3 — Шифрование текстового файла

В качестве тестового архива был взят rar архив с кодом текущей и предыдущих лабораторных работы.

На рисунке 2.5 представлен процесс шифровки и расшифровки указанного архива.

На рисунках 2.6- 2.8 представлены попытки открытия архивных файлов.

```

~/Projects/InfoSecurity main* ↑
>
• bat --line-range :1 test.enc.txt -A


|   |                                                                                    |
|---|------------------------------------------------------------------------------------|
|   | File: test.enc.txt <BINARY>                                                        |
| 1 |  |


```

~/Projects/InfoSecurity main* ↑
>
• go run ./cmd/aes/crypt/main.go -in test.enc.txt -out test.dec.txt -op decrypt
~/Projects/InfoSecurity main* ↑
>
• bat --line-range :5 test.dec.txt

| | |
|---|---|
| | File: test.dec.txt |
| 1 | 1. 2. Цель юнит-тестирования |
| 2 | Прежде чем углубляться в тему юнит-тестирования, давайте рассмотрим, для чего |
| 3 | всёобод нужно юнит-тестирование и какой цели оно помогает добиться. Считается, |
| 4 | что юнит-тестирование улучшает качество кода проекта. И это правда: необходимости |
| 5 | написания юнит-тестов обычно приводят к улучшению качества кода. Но |


```

~/Projects/InfoSecurity main* ↑

```


```


```

Рисунок 2.4 — Расшифровка текстового файла

```

• bat --line-range :1 code.rar -A


|   |                                                                                      |
|---|--------------------------------------------------------------------------------------|
|   | File: code.rar <BINARY>                                                              |
| 1 |  |


```

~/Projects/InfoSecurity main* ↑
>
• go run ./cmd/aes/crypt/main.go -in code.rar -out code.enc.rar -op encrypt
~/Projects/InfoSecurity main* ↑
>
• bat --line-range :1 code.rar -A

| | |
|---|--|
| | File: code.rar <BINARY> |
| 1 |  |


```

~/Projects/InfoSecurity main* ↑
>
• go run ./cmd/aes/crypt/main.go -in code.enc.rar -out code.dec.rar -op decrypt
~/Projects/InfoSecurity main* ↑
>
• bat --line-range :1 code.rar -A


|   |                                                                                      |
|---|--------------------------------------------------------------------------------------|
|   | File: code.rar <BINARY>                                                              |
| 1 |  |


```


```


```

Рисунок 2.5 — Шифрование rar файла

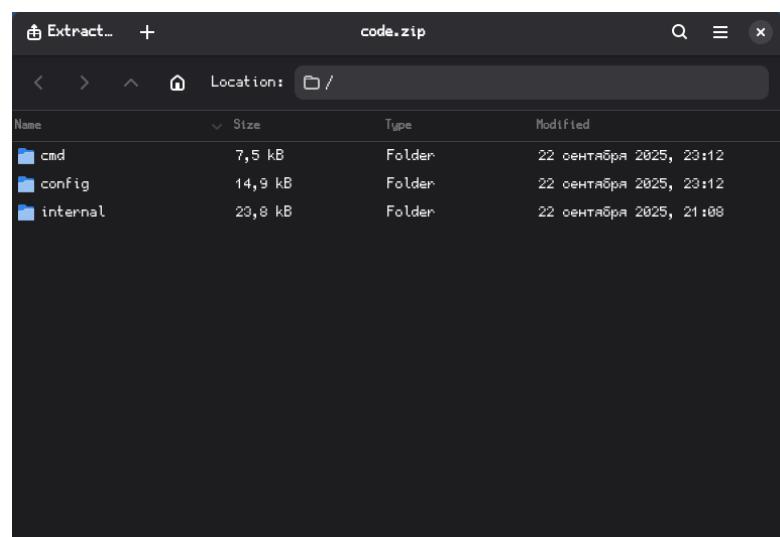


Рисунок 2.6 — Открытие исходного архива

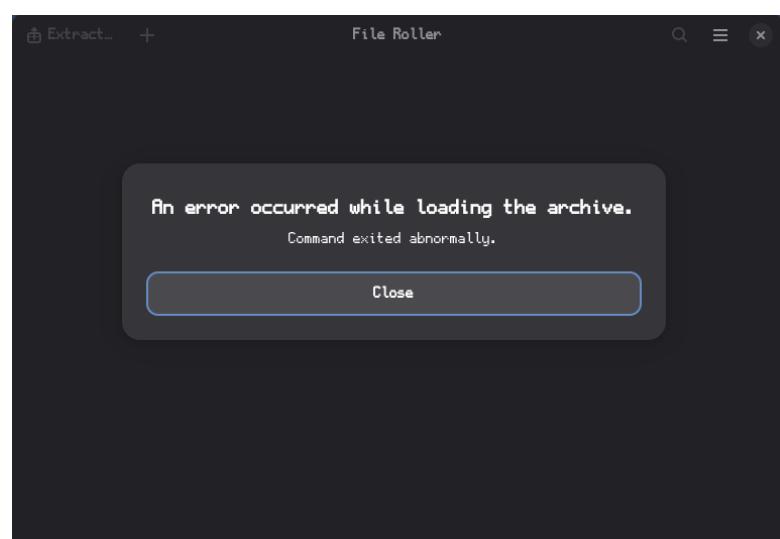


Рисунок 2.7 — Попытка открытия зашифрованного архива

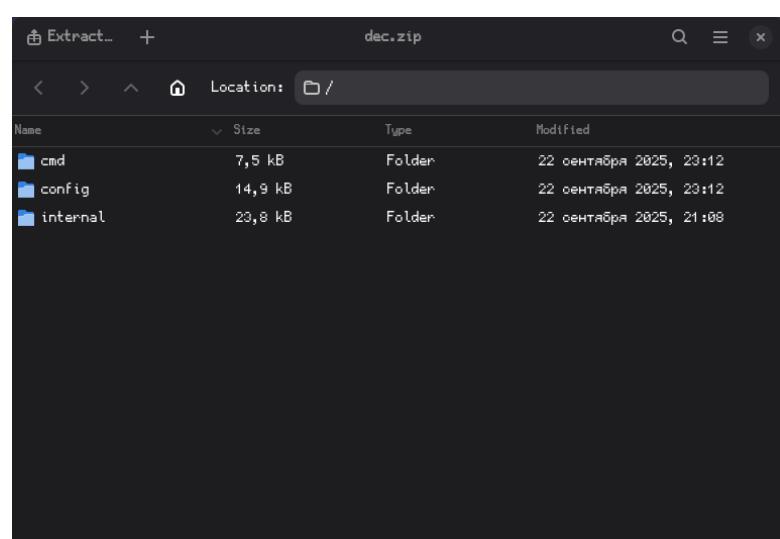


Рисунок 2.8 — Открытие расшифрованного архива

ЗАКЛЮЧЕНИЕ

В результате лабораторной работы была разработана программная реализация алгоритма AES.

Были выполнены следующие задачи:

- 1) проанализирован принцип работы алгоритма AES;
 - 2) реализованы описанные алгоритмы шифрования и расшифровки;
 - 3) проведено шифрование и расшифровка текстового файла и rar-архива;
- Все поставленные цели и задачи были выполнены.