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 №4
по курсу "Защита иформации"

Тема Алгоритм шифрования DES

Студент Нисуев Н. Ф.

Группа ИУ7-72Б

Преподаватель Руденкова Ю.С.

Москва — 2025 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Симметричное шифрование использует один и тот же ключ для шифрования и расшифрования данных, что обеспечивает высокую скорость обработки, но требует безопасного способа передачи ключа. Шифровальная алгоритм DES — алгоритм, разработанный в 1977 году компанией IBM и являющийся официальным стандартом шифрования.

Цель лабораторной работы: проектирование и разработка программной реализации алгоритма шифрования DES. Для достижения поставленной цели необходимо выполнить следующие задачи:

— провести анализ работы алгоритма DES;

— описать алгоритм шифрования с открытым ключом;

— реализовать и протестировать реализацию алгоритма шифрования.

1 Аналитическая часть

В этом разделе будут рассмотрены, виды симметричного шифрования, алгоритмы перестановки и подстановки, криптографический алгоритм DES.

1.1 Виды симметричного шифрования

Симметричное шифрование делится на два основных вида: поточное и блочное. В поточном шифровании данные обрабатываются побитово или посимвольно, в то время как в блочном шифровании данные разбиваются на блоки фиксированного размера.

— **Поточное шифрование** — шифрование происходит в реальном времени, каждый бит открытого текста шифруется независимо;

— **Блочное шифрование** — данные разбиваются на блоки фиксированной длины, и каждый блок шифруется отдельно.

1.2 Алгоритм DES

Шифровальная алгоритм DES (англ. *Data Encryption Standart* — DES) — симметричный шифровальный алгоритм, разработанный в 1977 году компанией IBM. Он использует блочное шифрование, длина блока фиксирована и равна 64 битам. Однако каждые 8 бит в ключе игнорируются, что приводит к правильной длине ключа 56 бит в DES. Однако в любом случае один блок на 64 бита является вечной организацией DES. Он состоит из 3 следующих шагов, рисунок 1.1:

— начальная перестановка (англ. *Initial Permutation* — IP), во время которой биты переставляются в порядке, определённом в специальной таблице;

— 16 раундов шифрования;

— завершающей перестановки (англ. *Final Permutation* – FP), соовершающей преобразования, обратные сделанным на первом шаге.

Раунд шифрования состоит из 5 следующих этапов

1) расширение (англ. *expansion* — E);

2) получение ключа раунда (англ. *Round Key* – RK);

3) скремблирование (англ. *substitution* — S);

4) перестановка (англ. *permutation* — P)

5) смешивание ключа (англ. *key mixing* — KM).

На рисунке 1.1 представлена обобщенная схема шифрования алгоритма DES.

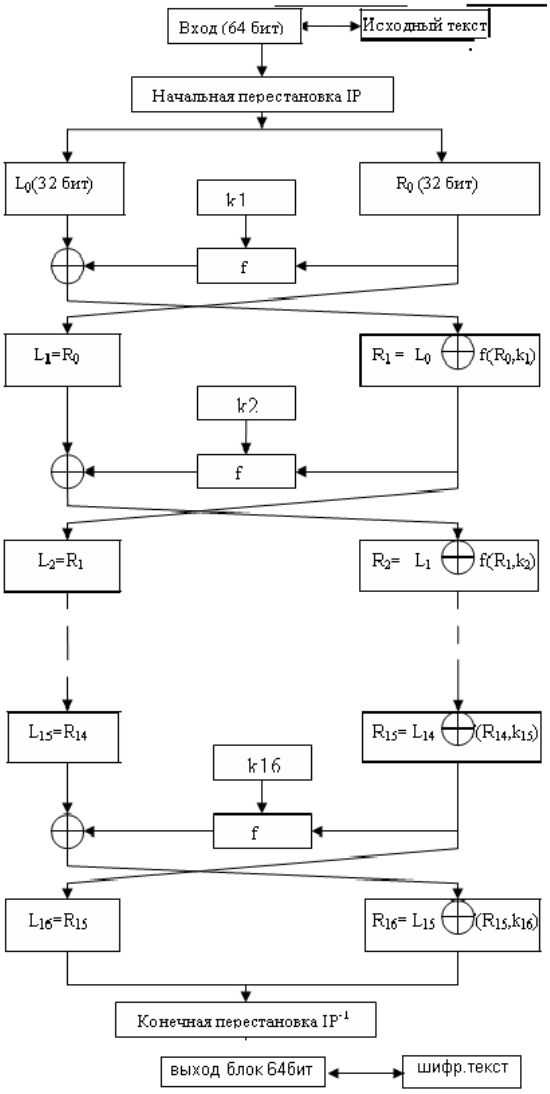

Рисунок 1.1 – Обобщенная схема шифрования в алгоритме DES

Расширение, во время которого каждая из половин блока шифрования по 32 бит дополняется путём перестановки и дублировоания бит до длины в 48 бит.

Получение ключа раунда необходимо для применения в раунде шифрования 48-битного ключа раунда, полученного из основного ключа DES. Основной ключ

имеет длину 64 бита, однако значащих бит из 64 всего 56, остальные добавлены для избыточности и контроля передачи ключа. Из этих 56 бит получают 48 путём разбиения на равные части и применению битовой операции циклического сдвига и нахождению нового значения посредством специальной таблицы.

Скремблирование предназначено для получения из 48-битного потока 32-битного путём разбиения на 6 частей по 8 бит и обработки каждой части в S-блоках (англ. *Substitution boxes*), которые заменяют блоки с длиной 6 бит на блоки 4 бит посредством использования специальной таблицы.

Перестановка представляет из себя перемешивания полученной последовательности из 32 бит при помощи таблицы перемешивания.

Смешивание ключа представляет из себя операцию XOR полученного 32-битного значения с ключом раунда.

1.3 Режимы работы алгоритма DES

Режим шифрования — метод применения блочного шифра, позволяющий преобразовать последовательность блоков открытых данных в последовательность блоков зашифрованных данных.

Для DES рекомендованы следующие режими работы:

1) режим электронной кодовой книги (англ. *Electronic Code Bloc* — ECB);
2) режим сцепления блоков (англ. *Cipher Block Chaining* – CBC);
3) режим параллельноого сцепления блоков (англ. *Parallel Cipher Block Chaining* – PCBC);
4) режим обратной связи по шифротексту (англ. *Cipher Feed Back* — CFB);
5) режим обратной связи по выходу (англ. *Output Feed Back* — OFB).

В данной работе будет электронной кодовой книги (ECB).

1.4 Алгоритмы перестановки и подстановки

Алгоритм перестановки — это метод шифрования, при котором биты входных данных перемешиваются в соответствии с фиксированной схемой. Порядок следования битов изменяется, но их значения остаются прежними. Пример: Initial Permutation в DES.

Алгоритм подстановки — это метод шифрования, при котором биты или группы битов заменяются на другие значения согласно заранее определенной таблице замен. Пример: S-блоки в DES;

Алгоритм DES использует оба подхода.

2 Технологическая часть

2.1 Средства реализации

Для программной реализации шифровальной машины был выбран язык Rust. В данном языке есть все требующиеся инструменты для данной лабораторной работы.

2.2 Реализация алгоритма

В листингах 2.1 — 2.2 представлена реализация алгоритма DES.

Листинг 2.1 – Генерация ключа

```
mod hex {
    pub fn encode(b: &[u8]) -> String {
        let mut s = String::with_capacity(b.len() * 2);
        for &byte in b {
            s.push(hex_char(byte >> 4));
            s.push(hex_char(byte & 0xF));
        }
        s
    }
}

pub fn genkey_file(keyfile: &Path) -> Result<(), String> {
    let mut key = [0u8; 8];
    OsRng.fill_bytes(&mut key);

    let hex = hex::encode(&key);

    let mut f = File::create(keyfile).map_err(|e| format!("Failed
       create keyfile: {}", e))?;
    f.write_all(hex.as_bytes())
        .map_err(|e| format!("Failed write keyfile: {}", e))?;

    Ok(())
}
```

Листинг 2.2 – Шифрование/расшифровка алгоритмом RSA

```
fn f_func(r: &[u8], k: &[u8]) -> Vec<u8> {
    let r_exp = bits::permute(r, &E);
    let x = bits::xor_bits(&r_exp, k);
    let s_out = sbox_substitute(&x);
    bits::permute(&s_out, &P)
}

fn des_block(block: &[u8; 8], subkeys: &Vec<Vec<u8>>, encrypt: bool) ->
   [u8; 8] {
    let bits = bits::bytes_to_bitvec(block);
    let ip = bits::permute(&bits, &IP);

    let mut l = ip[..32].to_vec();
    let mut r = ip[32..].to_vec();

    let keys_iter: Box<dyn Iterator<Item = &Vec<u8>>> = if encrypt {
        Box::new(subkeys.iter())
    } else {
        Box::new(subkeys.iter().rev())
    };

    for k in keys_iter {
        let new_l = r.clone();
        let f_out = f_function(&r, k);
        let new_r = bits::xor_bits(&l, &f_out);
        l = new_l;
        r = new_r;
    }

    let preout: Vec<u8> = r.into_iter().chain(l.into_iter()).collect();
    let final_bits = bits::permute(&preout, &FP);
    let out_bytes = bits::bitvec_to_bytes(&final_bits);

    let mut arr = [0u8; 8];
    arr.copy_from_slice(&out_bytes[..8]);
    arr
}

fn pkcs5_pad(data: &[u8]) -> Vec<u8> {
    let pad_len = 8 - (data.len() % 8);
```

```
40      let mut out = data.to_vec();
41
42      out.extend(std::iter::repeat(pad_len as u8).take(pad_len));
43      out
44  }
45
46  fn pkcs5_unpad(data: &[u8]) -> Result<Vec<u8>, String> {
47      if data.len() == 0 || data.len() % 8 != 0 {
48          return Err("Invalid padded data length".to_string());
49      }
50
51      let pad_len = *data.last().unwrap() as usize;
52      if pad_len < 1 || pad_len > 8 {
53          return Err("Invalid padding byte".to_string());
54      }
55
56      let end = data.len();
57      for &b in &data[end - pad_len..] {
58          if b as usize != pad_len {
59              return Err("Invalid padding contents".to_string());
60          }
61      }
62
63      Ok(data[..end - pad_len].to_vec())
64  }
65
66  pub fn encrypt_file(keyfile: &Path, infile: &Path, outfile: &Path) ->
        Result<(), String> {
67      let key = load_key(keyfile)?;
68      let subkeys = generate_subkeys(&key);
69
70      let plaintext = fs::read(infile).map_err(|e| format!("Failed read
            infile: {}", e))?;
71      let padded = pkcs5_pad(&plaintext);
72
73      let mut out = Vec::with_capacity(padded.len());
74      for chunk in padded.chunks(8) {
75          let mut arr = [0u8; 8];
76          arr.copy_from_slice(chunk);
77
78          let enc = des_block(&arr, &subkeys, true);
```

```
79          out.extend_from_slice(&enc);
80      }
81
82      let mut f = File::create(outfile).map_err(|e| format!("Failed
            create outfile: {}", e))?;
83      f.write_all(&out)
84          .map_err(|e| format!("Failed write outfile: {}", e))?;
85
86      Ok(())
87  }
88
89  pub fn decrypt_file(keyfile: &Path, infile: &Path, outfile: &Path) ->
        Result<(), String> {
90      let key = load_key(keyfile)?;
91      let subkeys = generate_subkeys(&key);
92
93      let ciphertext = fs::read(infile).map_err(|e| format!("Failed read
            infile: {}", e))?;
94      if ciphertext.len() % 8 != 0 {
95          return Err("Ciphertext length not multiple of 8".to_string());
96      }
97
98      let mut out = Vec::with_capacity(ciphertext.len());
99      for chunk in ciphertext.chunks(8) {
100         let mut arr = [0u8; 8];
101         arr.copy_from_slice(chunk);
102
103         let dec = des_block(&arr, &subkeys, false);
104         out.extend_from_slice(&dec);
105     }
106
107     let unp = pkcs5_unpad(&out).map_err(|e| format!("Unpad error: {}",
            e))?;
108
109     let mut f = File::create(outfile).map_err(|e| format!("Failed
            create outfile: {}", e))?;
110     f.write_all(&unp)
111         .map_err(|e| format!("Failed write outfile: {}", e))?;
112
113     Ok(())
114 }
```

2.3 Пример работы программы

На рисунке 2.1 представлен пример работы программы на текстовом фале.

```
PS D:\Programms\InfSecurity\lab_03> .\des.exe genkey
Key successfully saved in des.key
PS D:\Programms\InfSecurity\lab_03> cat .\testf\test.txt
lwnvjwnvjlwnvlnwjvnjwljewjviwjviewhvouwnvewonenviwbviwebvioewbviowe
PS D:\Programms\InfSecurity\lab_03> .\des.exe encrypt .\testf\test.txt -o etext.txt -k .\des.key
.\testf\test.txt successfully encrypted in file etext.txt
PS D:\Programms\InfSecurity\lab_03> cat .\etext.txt
♦.♦♦♦♦!♦ш♣p♦♦♦d♦♦u♦T♦!
,♦♦♦!4♦♦ f♦♦f♦91♦♦♦♦♦EhZ♦♦♦F;♦
♦Y♪♦
PS D:\Programms\InfSecurity\lab_03> .\des.exe decrypt .\etext.txt -o dtext.txt -k .\des.key
.\etext.txt successfully decrypted in file dtext.txt
PS D:\Programms\InfSecurity\lab_03> cat .\dtext.txt
lwnvjwnvjlwnvlnwjvnjwljewjviwjviewhvouwnvewonenviwbviwebvioewbviowe
PS D:\Programms\InfSecurity\lab_03>
```

Рисунок 2.1 – Пример работы программы на текстовом файле

На рисунках 2.2 — 2.5 представлен пример работы программы на zip-фале.

```
PS D:\Programms\InfSecurity\lab_03> .\des.exe encrypt .\test\example.zip -o .\test\ezip.zip -k .\des.key
.\test\example.zip successfully encrypted in file .\test\ezip.zip
PS D:\Programms\InfSecurity\lab_03> .\des.exe encrypt .\test\ezip.zip -o .\test\dzip.zip -k .\des.key
.\test\ezip.zip successfully encrypted in file .\test\dzip.zip
PS D:\Programms\InfSecurity\lab_03>
```

Рисунок 2.2 – Шифрация/дешифрация zip-файла

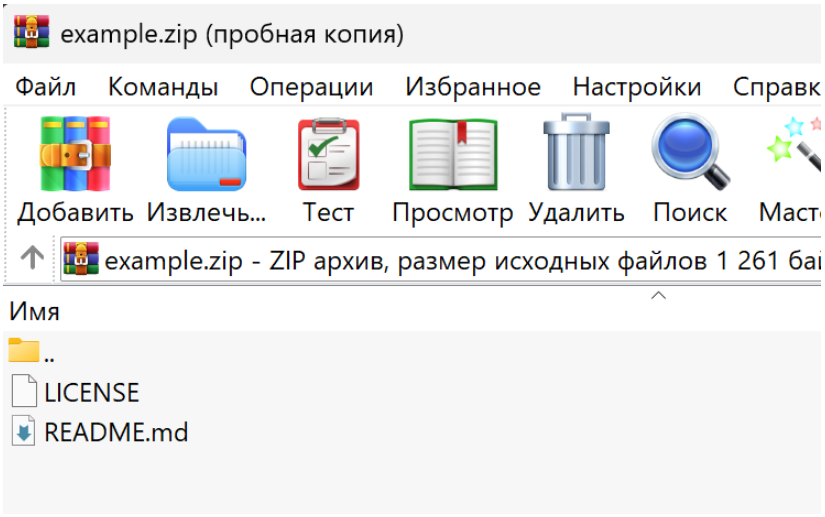

Рисунок 2.3 – Пример zip-файла

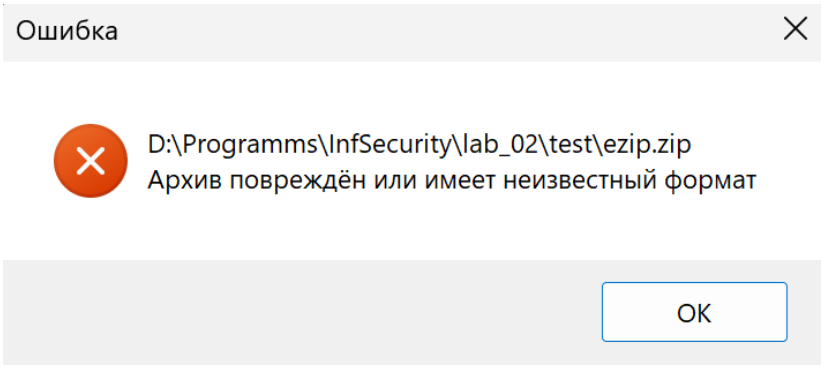

Рисунок 2.4 – Зашифрованный zip-файл

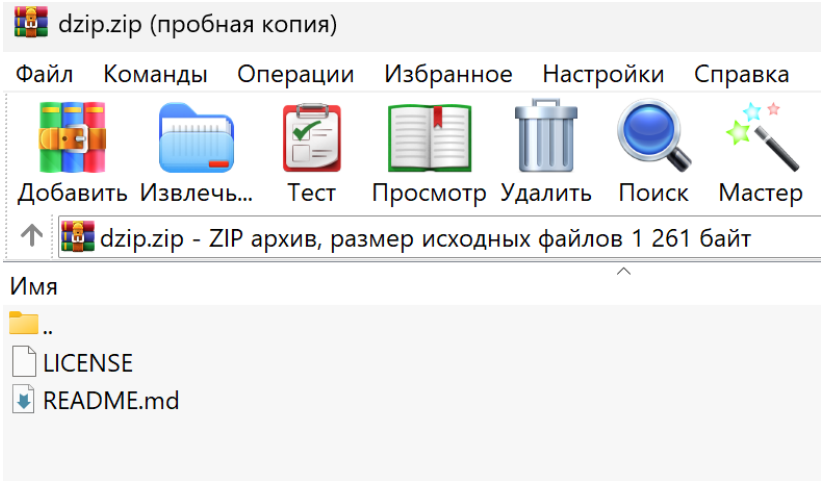

Рисунок 2.5 – Дешифрованный zip-файл

ЗАКЛЮЧЕНИЕ

В результате лабораторной работы был изучен алгоритма шифрования DES и разработана программная реализация. Были решены следующие задачи:

— проведен анализ работы алгоритма DES;

— описан алгоритм шифрования DES;

— реализована и протестирована реализация алгоритма шифрования DES.