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 курсу "Защита иформации"

Тема Алгоритм шифрования AES

Студент Нисуев Н. Ф.

Группа ИУ7-72Б

Преподаватель Руденкова Ю.С.

Москва — 2025 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Шифрование информации — занятие, которым человек занимался ещё до начала первого тысячелетия, занятие, позволяющее защитить информацию от посторонних лиц.

Шифровальная алгоритм AES — алгоритм, разработанный в 2001 году Национальным университетом стандартов и технологий США и пришедший на смену алгоритму AES.

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

Для достижения поставленной цели необходимо выполнить следующие задачи:

— провести анализ работы алгоритма AES и его режим работы CBC;

— описать алгоритм шифрования с открытым ключом;

— реализовать и протестировать реализацию алгоритма шифрования.

1 Аналитическая часть

В этом разделе будут рассмотрен шифровальный алгоритм AES в режиме шифрования CFB.

1.1 Алгоритм AES

Шифровальная алгоритм AES (англ. *Advanced Encryption Standart* — AES) — симметричный блочнный шифровальный алгоритм, разработанный в 2001 году Национальный институтом стандартов и технологий США. Он использует блочное шифрование, длина блока фиксирована и равна 128 битам, длина ключа 128, 192 либо же 256 бит. Он состоит раундов шифрования,количество которых зависит от длины ключа: 10 раундов для ключа размером 128 бит, 12 раундов для ключа размером 192 бита и 14 раундов для ключа размером 256 бит.

Прежде чем перейти к раундам шифрования, происходит генерация ключей раунда (раундовых ключей) из исходного ключа, Рассмотрим, как это происходит.

1.1.1 Получение ключей раунда

Определим фунцию g, изменяющую четырёхбайтовое слово так, как указано на рисунке 1.1.

Ключей раундов k_i необходимо на 1 больше, чем количество раундов, т.е. 11 ключей раундов для основногоключа длиной 128 бит, 13 ключей раунда для основного ключа длиной 192 бита и 15 ключей раунда для основного ключа длинной 256 бит.

Функция g:

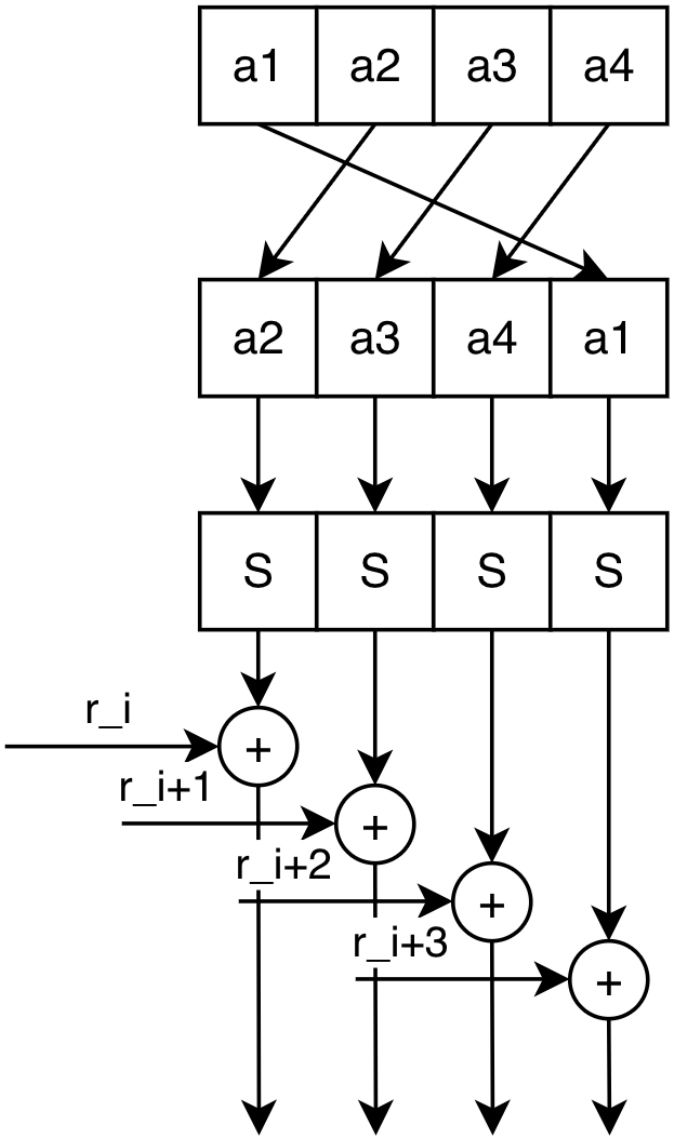

Рисунок 1.1 – Схема функции g

Алгоритм получения ключа раунда из исходного ключа преставлен в виде схемы алгоритма на рисунке 1.2.

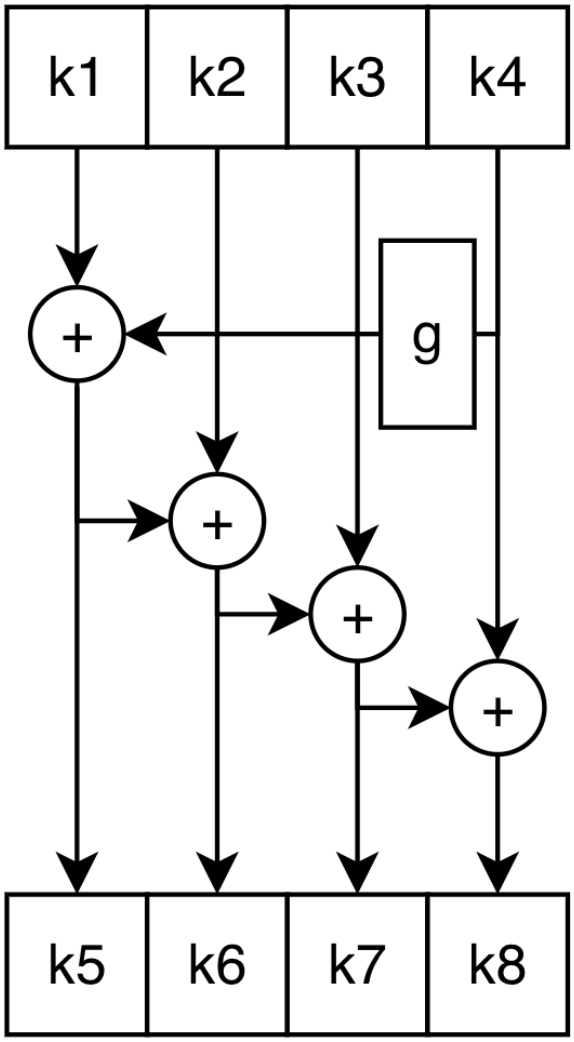

Рисунок 1.2 – Схема функции g

1.1.2 Раунд шифрования

Раунд шифрования состоит из 5 следующих этапов

1) замена (англ. *confussion*);
2) процедура перестановки строк (англ. *row-row mix procedure* — RR);
3) процедура перестановки столбцов (англ. *row-columns mix* — RC);
4) смешивание ключа (англ. *key mixing* — KM).

Замена обеспечивает нелинейность алгоритма шифрования, обрабатываая каждый байт состояния, производя нелинейную замену байт с использованием таблицы замен.

Процедура перестановки строк представляет из себя циклический сдвиг строки ссостояний на количество байт, зависящее от номера строки.

Процедура перестановки столбцов 4 байта каждого столбца смешиваются с использовоанием обратимой линейной трансформации. На последнем раунду эта процедура не выполняется.

Смешивание ключа представляет из себя операцию XOR с ключом раунда, полученным заранее.

1.2 Режимы работы алгоритма AES

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

Для AES 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 CBC.

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 – алгоритм AES128

```
1  mod consts;
2  use consts::*;
3
4  type State = [[u8; 4]; 4];
5
6  pub struct Aes128 {
7      round_keys: Vec<[u32; 4]>,
8  }
9
10 impl Aes128 {
11     pub fn new(key: &[u8; 16]) -> Self {
12         let round_keys = Self::key_expansion(key);
13         Self { round_keys }
14     }
15
16     fn key_expansion(key: &[u8; 16]) -> Vec<[u32; 4]> {
17         let mut round_keys = Vec::with_capacity(11);
18
19
20         let mut w: Vec<u32> = key.chunks(4)
21             .map(|chunk| u32::from_be_bytes(chunk.try_into().unwrap()))
22             .collect();
23
24
25         for i in 4..44 {
```

```
26              let mut temp = w[i - 1];
27
28              if i % 4 == 0 {
29                  temp = Self::sub_word(Self::rot_word(temp)) ^ RCON[i /
                        4 - 1];
30              }
31
32              w.push(w[i - 4] ^ temp);
33          }
34
35
36          for i in 0..11 {
37              round_keys.push([
38                  w[4 * i],
39                  w[4 * i + 1],
40                  w[4 * i + 2],
41                  w[4 * i + 3],
42              ]);
43          }
44
45          round_keys
46      }
47
48      fn sub_word(word: u32) -> u32 {
49          let bytes = word.to_be_bytes();
50          u32::from_be_bytes([
51              S_BOX[bytes[0] as usize],
52              S_BOX[bytes[1] as usize],
53              S_BOX[bytes[2] as usize],
54              S_BOX[bytes[3] as usize],
55          ])
56      }
57
58      fn rot_word(word: u32) -> u32 {
59          word.rotate_left(8)
60      }
61
62      pub fn encrypt(&self, input: &[u8; 16]) -> [u8; 16] {
63          let mut state = Self::bytes_to_state(input);
64
65
```

```
66          self.add_round_key(&mut state, 0);
67
68
69          for round in 1..10 {
70              self.sub_bytes(&mut state);
71              self.shift_rows(&mut state);
72              self.mix_columns(&mut state);
73              self.add_round_key(&mut state, round);
74          }
75
76          self.sub_bytes(&mut state);
77          self.shift_rows(&mut state);
78          self.add_round_key(&mut state, 10);
79
80          Self::state_to_bytes(&state)
81      }
82
83      pub fn decrypt(&self, input: &[u8; 16]) -> [u8; 16] {
84          let mut state = Self::bytes_to_state(input);
85
86          self.add_round_key(&mut state, 10);
87          self.inv_shift_rows(&mut state);
88          self.inv_sub_bytes(&mut state);
89
90          for round in (1..10).rev() {
91              self.add_round_key(&mut state, round);
92              self.inv_mix_columns(&mut state);
93              self.inv_shift_rows(&mut state);
94              self.inv_sub_bytes(&mut state);
95          }
96
97          self.add_round_key(&mut state, 0);
98
99          Self::state_to_bytes(&state)
100     }
101
102     fn bytes_to_state(bytes: &[u8; 16]) -> State {
103         let mut state = [[0u8; 4]; 4];
104         for i in 0..4 {
105             for j in 0..4 {
106                 state[j][i] = bytes[i * 4 + j];
```

```
107             }
108         }
109         state
110     }
111
112     fn state_to_bytes(state: &State) -> [u8; 16] {
113         let mut bytes = [0u8; 16];
114         for i in 0..4 {
115             for j in 0..4 {
116                 bytes[i * 4 + j] = state[j][i];
117             }
118         }
119         bytes
120     }
121
122     fn sub_bytes(&self, state: &mut State) {
123         for row in state.iter_mut() {
124             for byte in row.iter_mut() {
125                 *byte = S_BOX[*byte as usize];
126             }
127         }
128     }
129
130     fn inv_sub_bytes(&self, state: &mut State) {
131         for row in state.iter_mut() {
132             for byte in row.iter_mut() {
133                 *byte = INV_S_BOX[*byte as usize];
134             }
135         }
136     }
137
138     fn shift_rows(&self, state: &mut State) {
139         for i in 1..4 {
140             state[i].rotate_left(i);
141         }
142     }
143
144     fn inv_shift_rows(&self, state: &mut State) {
145         for i in 1..4 {
146             state[i].rotate_right(i);
147         }
```

```
148     }
149
150     fn mix_columns(&self, state: &mut State) {
151         for i in 0..4 {
152             let a = state[0][i];
153             let b = state[1][i];
154             let c = state[2][i];
155             let d = state[3][i];
156
157             state[0][i] = Self::gmul(0x02, a) ^ Self::gmul(0x03, b) ^ c
                    ^ d;
158             state[1][i] = a ^ Self::gmul(0x02, b) ^ Self::gmul(0x03, c)
                    ^ d;
159             state[2][i] = a ^ b ^ Self::gmul(0x02, c) ^
                    Self::gmul(0x03, d);
160             state[3][i] = Self::gmul(0x03, a) ^ b ^ c ^
                    Self::gmul(0x02, d);
161         }
162     }
163
164     fn inv_mix_columns(&self, state: &mut State) {
165         for i in 0..4 {
166             let a = state[0][i];
167             let b = state[1][i];
168             let c = state[2][i];
169             let d = state[3][i];
170
171             state[0][i] = Self::gmul(0x0e, a) ^ Self::gmul(0x0b, b) ^
                    Self::gmul(0x0d, c) ^ Self::gmul(0x09, d);
172             state[1][i] = Self::gmul(0x09, a) ^ Self::gmul(0x0e, b) ^
                    Self::gmul(0x0b, c) ^ Self::gmul(0x0d, d);
173             state[2][i] = Self::gmul(0x0d, a) ^ Self::gmul(0x09, b) ^
                    Self::gmul(0x0e, c) ^ Self::gmul(0x0b, d);
174             state[3][i] = Self::gmul(0x0b, a) ^ Self::gmul(0x0d, b) ^
                    Self::gmul(0x09, c) ^ Self::gmul(0x0e, d);
175         }
176     }
177
178     fn gmul(a: u8, b: u8) -> u8 {
179         let mut p = 0;
180         let mut a = a;
```

```
181         let mut b = b;
182
183         for _ in 0..8 {
184             if b & 1 != 0 {
185                 p ^= a;
186             }
187
188             let hi_bit_set = a & 0x80 != 0;
189             a <<= 1;
190             if hi_bit_set {
191                 a ^= 0x1b;
192             }
193             b >>= 1;
194         }
195
196         p
197     }
198
199     fn add_round_key(&self, state: &mut State, round: usize) {
200         let round_key = &self.round_keys[round];
201
202         for i in 0..4 {
203             let key_bytes = round_key[i].to_be_bytes();
204             for j in 0..4 {
205                 state[j][i] ^= key_bytes[j];
206             }
207         }
208     }
209 }
```

Листинг 2.2 – алгоритм AES128 с режимом работы CBC

```
1 mod aes128;
2 pub mod keygen;
3
4 use aes128::Aes128;
5
6 pub struct Aes128Cbc {
7     aes: Aes128,
8     iv: [u8; 16],
9 }
10
```

```
impl Aes128Cbc {
    pub fn new(key: &[u8; 16], iv: &[u8; 16]) -> Self {
        Self {
            aes: Aes128::new(key),
            iv: *iv,
        }
    }

    pub fn encrypt(&self, plaintext: &[u8]) -> Vec<u8> {
        let mut ciphertext = Vec::new();
        let mut prev_block = self.iv;

        for chunk in plaintext.chunks(16) {
            let mut block = [0u8; 16];
            let chunk_len = chunk.len();
            block[..chunk_len].copy_from_slice(chunk);

            if chunk_len < 16 {
                let pad_byte = (16 - chunk_len) as u8;
                for i in chunk_len..16 {
                    block[i] = pad_byte;
                }
            }

            for i in 0..16 {
                block[i] ^= prev_block[i];
            }

            let encrypted_block = self.aes.encrypt(&block);
            ciphertext.extend_from_slice(&encrypted_block);
            prev_block = encrypted_block;
        }

        ciphertext
    }

    pub fn decrypt(&self, ciphertext: &[u8]) -> Vec<u8> {
        let mut plaintext = Vec::new();
        let mut prev_block = self.iv;

        for chunk in ciphertext.chunks(16) {
```

```
52              let block: [u8; 16] = chunk.try_into().unwrap();
53              let decrypted_block = self.aes.decrypt(&block);
54
55              let mut plain_block = [0u8; 16];
56              for i in 0..16 {
57                  plain_block[i] = decrypted_block[i] ^ prev_block[i];
58              }
59
60              plaintext.extend_from_slice(&plain_block);
61              prev_block = block;
62          }
63
64          if let Some(&last_byte) = plaintext.last() {
65              let pad_len = last_byte as usize;
66              if pad_len <= 16 {
67                  plaintext.truncate(plaintext.len() - pad_len);
68              }
69          }
70
71          plaintext
72      }
73  }
```

2.3 Пример работы программы

На рисунке 2.1 представлен пример работы программы на текстовом фале.

```
PS D:\Programms\InfSecurity\lab_04> cargo run -- genkey
   Compiling aes v0.1.0 (D:\Programms\InfSecurity\lab_04)
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 1.22s
     Running `target\debug\aes.exe genkey`
Key successfully saved in .aes.key
PS D:\Programms\InfSecurity\lab_04> cargo run -- encrypt .\example.txt -k .\.aes.key
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.09s
     Running `target\debug\aes.exe encrypt .\example.txt -k .\.aes.key`
File '.\example.txt' successfully encrypted to '.\encrypted_example.txt'
PS D:\Programms\InfSecurity\lab_04> cargo run -- decrypt .\encrypted_example.txt -k .\.aes.key
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.07s
     Running `target\debug\aes.exe decrypt .\encrypted_example.txt -k .\.aes.key`
File '.\encrypted_example.txt' successfully encrypted to '.\decrypted_encrypted_example.txt'
PS D:\Programms\InfSecurity\lab_04> cat .\example.txt
mmwnvwoehvioewbvouwbvouwbvuoebewuovbewoubveowbvobwoibo
PS D:\Programms\InfSecurity\lab_04> cat .\encrypted_example.txt
♦ʕ♦♦ź+_♦\
♦>♦7♦♦.♦_♦♦b♦pJ♦
♦=♦Pc♦♦♦yWd=)♦♦♦♦♦f♦D♦♦♦G♦♦
PS D:\Programms\InfSecurity\lab_04> cat .\decrypted_encrypted_example.txt
mmwnvwoehvioewbvouwbvouwbvuoebewuovbewoubveowbvobwoibo
PS D:\Programms\InfSecurity\lab_04>
```

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_04> cargo run -- encrypt .\example.rar -k .\.aes.key
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.09s
     Running `target\debug\aes.exe encrypt .\example.rar -k .\.aes.key`
File '.\example.rar' successfully encrypted to '.\encrypted_example.rar'
PS D:\Programms\InfSecurity\lab_04> cargo run -- decrypt .\encrypted_example.rar -k .\.aes.key
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.07s
     Running `target\debug\aes.exe decrypt .\encrypted_example.rar -k .\.aes.key`
File '.\encrypted_example.rar' successfully encrypted to '.\decrypted_encrypted_example.rar'
```

Рисунок 2.2 – Шифрация/дешифрация zip-файла

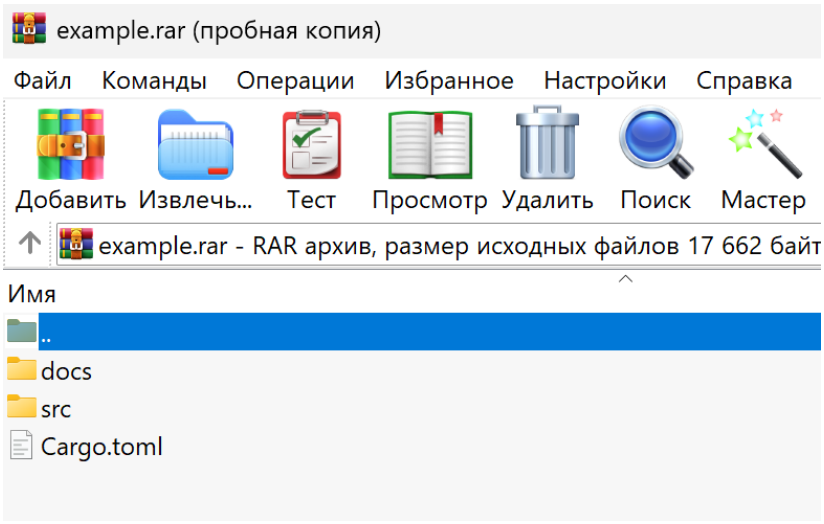

Рисунок 2.3 – Пример zip-файла

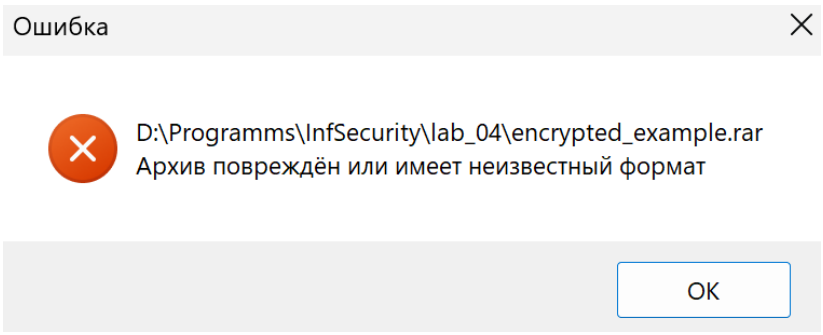

Рисунок 2.4 – Зашифрованный zip-файл

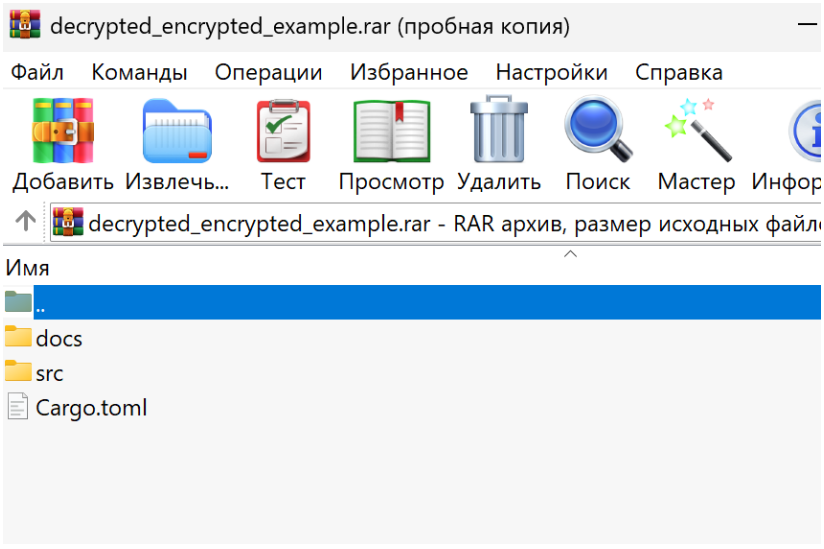

Рисунок 2.5 – Дешифрованный zip-файл

ЗАКЛЮЧЕНИЕ

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

— проведен анализ работы алгоритма AES;

— описан алгоритм шифрования AES;

— реализована и протестирована реализация алгоритма шифрования AES.