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 №2
по курсу "Защита иформации"

Тема Шифровальная машина Энигма

Студент Нисуев Н. Ф.

Группа ИУ7-72Б

Преподаватель Руденкова Ю.С.

Москва — 2025 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данного лаборотоной работы является проектирование и разработка программную реализацию машины "Энигмы".

Чтобы достигнуть поставленной цели, требуется решить следующие задачи:

— провести анализ работы шифровальной машины "Энигмы";

— описать алгоритм шифрования;

— реализовать алгоритм шифрования.

1 Аналитическая часть

1.1 Определения

Информация — сведения (сообщения, данные) независимо от формы их представления (накопленный опыт человечества).

Защита информации — принятие мер

— нормативно-правовых (законы),

— организационно-структурных (внутренние правила организации, направленные на людей),

— технических (программные и аппаратные средства, физическая защита),

направленных на:

1. предотвращение неправомерных действий в отношении информации:
   — доступ
   — копирование
   — модифицирование (изменение)
   — блокирование
   — предоставление (определенный круг лиц)
   — распространение (неопределенный круг лиц)
   — уничтожение (удаление)
2. соблюдение конфиденциальности информации ограниченного доступа,
3. реализацию права на доступ к информации.

Актив — все, что имеет ценность для субъекта и находится в его распоряжении.

Информационная сфера:

— информация,

— информационная инфраструктура (SW, HW, коммуникации),

— субъекты, обрабатывающих информацию,

— процедуры (что делаем),

— система регулирования отношений (что, где, кем, как) — как управлять.

Угроза — опасность, предполагающая возможность потерь (ущерба).

Безопасность — состояние защищенности интересов (целей) в условиях угроз.

Информационная безопасность — безопасность в условиях угроз в информационной сфере

Шифровальная машина "Энигма" — переносная электромеханическая машина, в которой электрическая схема меняется для каждой следующей буквы (многоалфавитный алгоритм)

Одноалфавитная подстановка — подстановка, при которой каждый символ открытого текста заменяется на некоторый, фиксированный при данном ключе символ того же алфавита. Все варианты одноалфавитной подстановки букв сводятся к замене по формуле: C(s)=(A(s)+B) mod D, где A-множитель, B-сдвиг, D-длина алфавита.

Многоалфавитные подстановки — маскируют естественную частотную статистику языка. Для каждой буквы алфавита строится массив символов замены такие, что:

1. ни одна пара массивов не пересекается, т.е. не содержит одинаковых элементов.

2. количество символов замены в каждом массиве пропорционально частоте появления буквы в открытом тексте.

Скрывается частота появления символа.

1.2 Алгоритм шифрования Энигмы

Шифровальная машина "Энигма" — переносная электромеханическая машина, в которой электрическая схема меняется для каждой следующей буквы (многоалфавитный алгоритм).

Шифровальная машина «Энигма» внешне выглядит как печатающая машинка, за исключением того факта, что шифруемые символы не печатаются автоматически на определённый лист бумаги, а указываются на панели посредством загорания лампочки.

Шифровальная машина «Энигма» обладает тремя основными механизмами.

1. Роторы — сердце всех шифровальных машин, которое со стороны классической криптографии они реализуют полиалфавитный алгоритм шифрования, а их определённо выстроенная позиция представляет собой один из основных ключей шифрования. Каждый ротор не эквивалентен другому ротору, потому как обладает своей специфичной настройкой. Военным на выбор давалось пять роторов, три из которых они вставляли в «Энигму».

2. Рефлектор — статичный механизм, позволяющий шифровальным машинам типа «Энигма» не вводить помимо операции шифрования дополнительную операцию расшифрования. Связано это с тем, что в терминологии классической криптографии рефлектор представляет собой просто частный случай моноалфавитного шифра.

3. Коммутатор — механизм, позволяющий оператору шифровальной машины варьировать содержимое проводов, попарно соединяющих буквы английского алфавита.

Перед расшифровкой роторы возвращаются в начальное состояние. Расшифровка идентична шифрованию.

На рисунке 1.1 представлена визуализация работы шифровальной машины "Энигма"на алфавите из восьми символов — $\{A, B, C, D, E, F, G, H\}$.

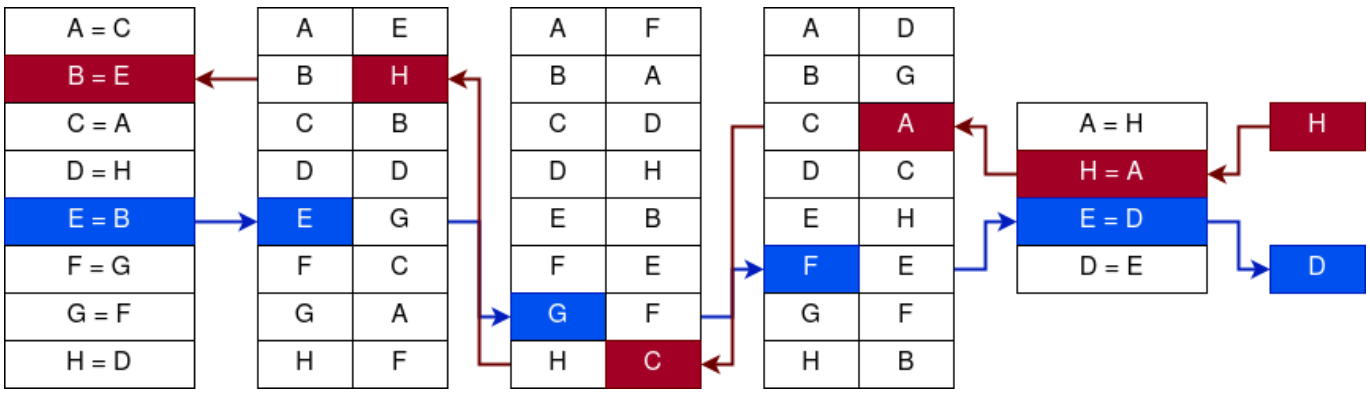

Рисунок 1.1 – Пример работы шифровальной машины "Энигма"

2 Конструкторская часть

2.1 Схема алгоритма

Схема алгоритма Энигмы представлена на рисунке 2.1.

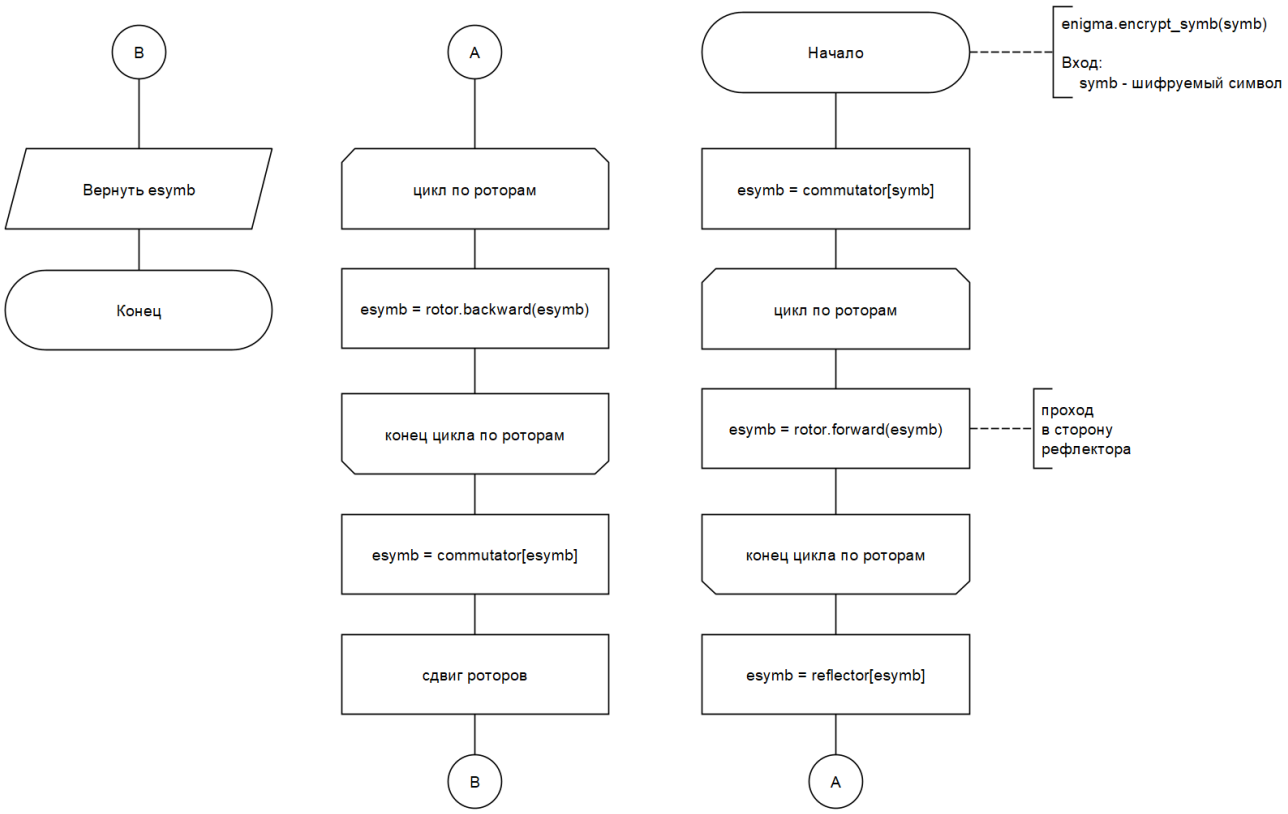

Рисунок 2.1 – Схема алгоритма шифровальной машины "Энигма"

3 Технологическая часть

3.1 Средства реализации

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

3.2 Реализация алгоритма

В листингах 3.1 — 3.3 представлена реализация шифровальной машины Энигма.

Листинг 3.1 – реализация рефлектора

```
pub struct Reflector<T> {
    alphabet: Vec<T>,
}

impl<T: Clone + Eq> Reflector<T> {
    pub fn from_alphabet(alphabet: &[T]) -> Result<Self, &str> {
        if alphabet.len() % 2 != 0 {
            return Err("Error: Can't be odd alphabet");
        }
        let mut rng = rng();
        let mut cipher = alphabet.to_vec();
        cipher.shuffle(&mut rng);

        Ok(Reflector { alphabet: cipher })
    }

    pub fn from_config(config: &[T]) -> Result<Self, &str> {
        if config.len() % 2 != 0 {
            return Err("Error: Can't be odd alphabet");
        }

        Ok(Reflector {
            alphabet: config.to_vec(),
        })
```

```
25      }
26
27      pub fn get_config(&self) -> Vec<T> { self.alphabet.clone() }
28
29      pub fn reflect(&self, input: &T) -> Option<T> {
30          let index = self.alphabet.iter().position(|x| x == input);
31
32          index.map(|i| self.alphabet[if i % 2 == 0 { i + 1 } else { i -
                1 }].clone())
33      }
34  }
```

Листинг 3.2 – реализация ротора

```
1   pub struct Rotor<T> {
2       position: usize,
3       alphabet_len: usize,
4
5       forward_alphabet: Vec<T>,
6       backward_alphabet: Vec<T>,
7   }
8
9   impl<T: Clone + Ord> Rotor<T> {
10      pub fn from_alphabet(alphabet: &[T]) -> Self {
11          let mut rng = rng();
12
13          let mut sorted_alphabet = alphabet.to_vec();
14          let mut shuffeled_alphabet = alphabet.to_vec();
15
16          sorted_alphabet.sort();
17          shuffeled_alphabet.shuffle(&mut rng);
18
19          Rotor {
20              alphabet_len: alphabet.len(),
21              forward_alphabet: sorted_alphabet,
22              backward_alphabet: shuffeled_alphabet,
23              position: 0,
24          }
25      }
26
27      pub fn from_config(config: &[T]) -> Self {
28          let mut sorted_alphabet = config.to_vec();
```

```
29          sorted_alphabet.sort();
30
31          Rotor {
32              alphabet_len: config.len(),
33              forward_alphabet: sorted_alphabet,
34              backward_alphabet: config.to_vec(),
35              position: 0,
36          }
37      }
38
39      pub fn get_config(&self) -> Vec<T> {
40          self.backward_alphabet.clone()
41      }
42
43      pub fn forward(&self, input: &T) -> Option<T> {
44          let index = self.forward_alphabet.iter().position(|x| x ==
                input);
45
46          index.map(|i| self.backward_alphabet[(i + self.position) %
                self.alphabet_len].clone())
47      }
48
49      pub fn backward(&self, input: &T) -> Option<T> {
50          let index = self.backward_alphabet.iter().position(|x| x ==
                input);
51
52          index.map(|i| {
53              self.forward_alphabet[(i + self.alphabet_len -
                    self.position) % self.alphabet_len]
54                  .clone()
55          })
56      }
57
58      pub fn is_at_init_position(&self) -> bool { self.position == 0 }
59
60      pub fn rotate(&mut self) {
61          self.position = (self.position + 1) % self.alphabet_len
62      }
63
64      pub fn reset(&mut self) { self.position = 0 }
65  }
```

Листинг 3.3 – реализация Энигмы

```rust
pub struct Enigma<T> {
    commutator: Option<Reflector<T>>,
    reflector: Reflector<T>,
    rotors: Vec<Rotor<T>>,
}

impl<T: Clone + Eq + Ord> Enigma<T> {
    pub fn from_alphabet(
        alphabet: &[T],
        rotors_cnt: u8,
        with_commutator: bool,
    ) -> Result<Self, &str> {
        let commutator = if with_commutator {
            Some(Reflector::from_alphabet(alphabet)?)
        } else {
            None
        };

        let reflector = Reflector::from_alphabet(alphabet)?;
        let rotors = (0..rotors_cnt)
            .map(|_| Rotor::from_alphabet(alphabet))
            .collect();

        Ok(Enigma {
            commutator,
            reflector,
            rotors,
        })
    }

    pub fn from_config<'a>(
        commutator_config: Option<&'a [T]>,
        reflector_config: &'a [T],
        rotors_configs: &'a [Vec<T>],
    ) -> Result<Self, &'a str> {
        let commutator = if let Some(cfg) = commutator_config {
            Some(Reflector::from_config(cfg)?)
        } else {
            None
        };
```

```
41
42          let reflector = Reflector::from_config(reflector_config)?;
43          let rotors = (0..rotors_configs.len())
44              .map(|i| {
45                  if i == 0 {
46                      Rotor::from_config(&rotors_configs[i])
47                  } else if rotors_configs[i - 1].len() ==
                        rotors_configs[i].len() {
48                      Rotor::from_config(&rotors_configs[i])
49                  } else {
50                      panic!("Different sizes of rotor configs")
51                  }
52              })
53              .collect();
54
55          Ok(Enigma {
56              commutator,
57              reflector,
58              rotors,
59          })
60      }
61
62      pub fn get_config(&self) -> (Option<Vec<T>>, Vec<T>, Vec<Vec<T>>) {
63          (
64              self.commutator.as_ref().map(|c| c.get_config()),
65              self.reflector.get_config(),
66              self.rotors.iter().map(|rotor|
                    rotor.get_config()).collect(),
67          )
68      }
69
70      fn encrypt_symbol(&mut self, symbol: &T) -> Result<T, &'static str>
            {
71          let mut encrypt_symb = symbol.clone();
72
73          if let Some(commutator) = &self.commutator {
74              encrypt_symb = commutator
75                  .reflect(&encrypt_symb)
76                  .ok_or("Symbol not in alphabet")?;
77          }
78
```

```
79          for rotor in &self.rotors {
80              encrypt_symb = rotor
81                  .forward(&encrypt_symb)
82                  .ok_or("Symbol not in alphabet")?;
83          }
84
85          encrypt_symb = self
86              .reflector
87              .reflect(&encrypt_symb)
88              .ok_or("Symbol not in alphabet")?;
89
90          for rotor in self.rotors.iter().rev() {
91              encrypt_symb = rotor
92                  .backward(&encrypt_symb)
93                  .ok_or("Symbol not in alphabet")?;
94          }
95
96          if let Some(commutator) = &self.commutator {
97              encrypt_symb = commutator
98                  .reflect(&encrypt_symb)
99                  .ok_or("Symbol not in alphabet")?;
100         }
101
102         self.rotate_rotors();
103
104         Ok(encrypt_symb.clone())
105     }
106
107     pub fn encrypt(&mut self, buf: &[T]) -> Result<Vec<T>, (usize,
        &'static str)> {
108         let mut ebuf = Vec::with_capacity(buf.len());
109
110         for (i, symb) in buf.iter().enumerate() {
111             ebuf.push(self.encrypt_symbol(symb).map_err(|err_str| (i,
                err_str))?);
112         }
113
114         Ok(ebuf)
115     }
116
117     pub fn decrypt(&mut self, buf: &[T]) -> Result<Vec<T>, (usize,
```

```
        &'static str)> {
118         self.encrypt(buf)
119     }
120
121     fn rotate_rotors(&mut self) {
122         for i in 0..self.rotors.len() {
123             if i == 0 {
124                 self.rotors[i].rotate();
125             } else if self.rotors[i - 1].is_at_init_position() {
126                 self.rotors[i].rotate();
127             }
128         }
129     }
130
131     pub fn reset(&mut self) {
132         for rotor in &mut self.rotors {
133             rotor.reset();
134         }
135     }
136 }
```

3.3 Пример работы программы

```
Электронный аналог шифровальной машины "Энигма"

Usage: enigma.exe [OPTIONS] <FILENAME>

Arguments:
  <FILENAME>  Имя шифруемого файла (обязательный)

Options:
  -c, --config <FILE>     Имя конфигурационного файла рефлектора и роторов
  -o, --out <FILE>        Имя зашифрованного выходного файла [default: e<FILENAME>]
  -n, --rotors-num <NUM>  Количество роторов (0-255) [default: 3]
  -m, --with-commutator   Использовать коммутатор (панель подключений)
  -h, --help              Print help (see more with '--help')
  -V, --version           Print version
```

Рисунок 3.1 – Флаги CLI

```
PS D:\Programms\InfoSecurity\lab_01> cargo run -- .\test_data\testfile.txt -n 3 -m -o e.txt
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.10s
     Running `target\debug\enigma.exe .\test_data\testfile.txt -n 3 -m -o e.txt`
Зашифрованные данные сохранены в файл e.txt
Введите файл в который сохранить конфигурацию Энигмы (./enigma.conf):
Сохранение в файл: ./enigma.conf
PS D:\Programms\InfoSecurity\lab_01> cargo run -- .\e.txt -c .\enigma.conf -o ./d.txt
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.09s
     Running `target\debug\enigma.exe .\e.txt -c .\enigma.conf -o ./d.txt`
Зашифрованные данные сохранены в файл ./d.txt
PS D:\Programms\InfoSecurity\lab_01> cat .\test_data\testfile.txt
nvnvjwnvwjnvowonvoiwnvuobvueqbvuq
vneaklnvoenovivnieo
bcibwicbwiub
PS D:\Programms\InfoSecurity\lab_01> cat .\e.txt
ZW`7PG,R^rV~C9@T|
lX0Wf7"
9
PS D:\Programms\InfoSecurity\lab_01> cat .\d.txt
nvnvjwnvwjnvowonvoiwnvuobvueqbvuq
vneaklnvoenovivnieo
bcibwicbwiub
```

Рисунок 3.2 – Пример работы программы

ЗАКЛЮЧЕНИЕ

В результате лабораторной работы были изучены принципы работы шифровальной машины "Энигма была реализована программа, способная шифровать и дешифровать файлы любого формата. Были решены следующие задачи:

— проведен анализ работы шифровальной машина "Энигма";

— описан алгоритм шифрования;

— реализован описанный алгоритм;