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 дисциплине «Защита информация»

Тема Шифровальная машина «Энигма»

Студент Шахнович Дмитрий Сергеевич

Группа ИУ7-72Б

Преподаватель Руденкова Ю.С.

Москва, 2025

Содержание

ВВЕДЕНИЕ

Цель работы: Разработка электронного аналога шифровальной машины. Шифрование и расшифровка архивного файла ы **Задачи:**

1) проанализировать принцип работы шифровальной машины «Энигма»;
2) составить схема алгоритма шифровальной машины «Энигма»;
3) реализовать описанный алгоритм шифрования;
4) провести шифрование и расшифровку архивного файла;

1 Аналитическая часть

1.1 Алгоритм шифровальной машины «Энигма»

Шифровальная машина Энигма – портативная электромеханическая шифровальная машина, разработанная и широко использовавшаяся в XX веке, прежде всего нацистской Германией во время Второй мировой войны для защиты служебной связи. Алгоритм Энигма относится к многоалфавитным, так как при повороте ротора меняется алфавит замены, а значит одни и те же символы будут зашифрованы разными символами, в зависимости от позиции в тексте.

Шифровальная машина «Энигма» шифрует исходный текст посимвольно за счёт трёх видов механизмов:

1) **Роторы** – основа машины «Энигма», который представляет из себя механизм, реализующий многоалфавитный алгоритм шифрования. В машину «Энигма» вставлялись три ротора. Ротор имеет два основных параметра: алфавит и место засечки. После шифровании очередного символа первый ротор поворачивается на одну позицию, за счёт чего меняется алфавит шифрования. Если при этом ротор проходит позицию засечки, то он «зацепляет» следующий ротор и тот также поворачивается на одну позицию.
2) **Рефлектор** – статичный механизм, который представляет из одноалфавитный подстановку, при этом такую, что если символ 1 заменяется на символ 2, то и символ 2 заменяется на символ 1, т. е. рефлектор коммутативен относительно подстановки. За счёт этого механизма машина «Энигма» может быть использована как для шифрования, так и для дешифровки, главное чтобы была одинаковая конфигурация в обоих случаях.
3) **Коммутатор** – статичный механизм, который позволяет попарно менять два символа перед подачей их на роторы механизма.

На рисунке 1.1 представлен пример работы машина «Энигма» без коммутатора для символов латинского алфавита, а на рисунке 1.2 с коммутатором для символов {A, B, C, D, E, F, G, H}.

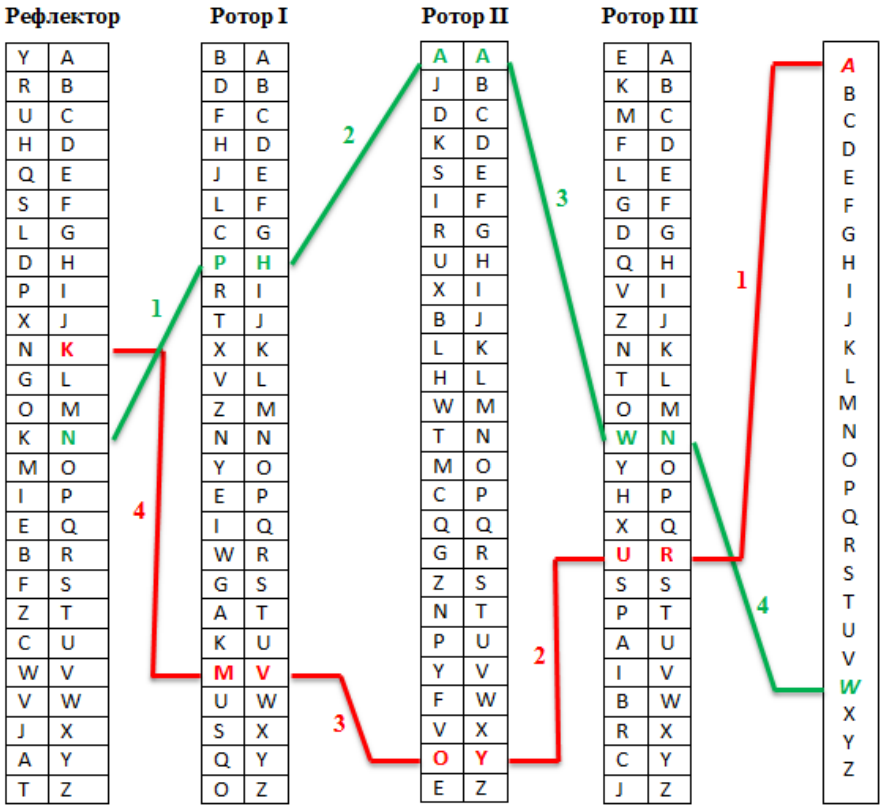

Рисунок 1.1 — Пример работы машина «Энигма» без коммутатора

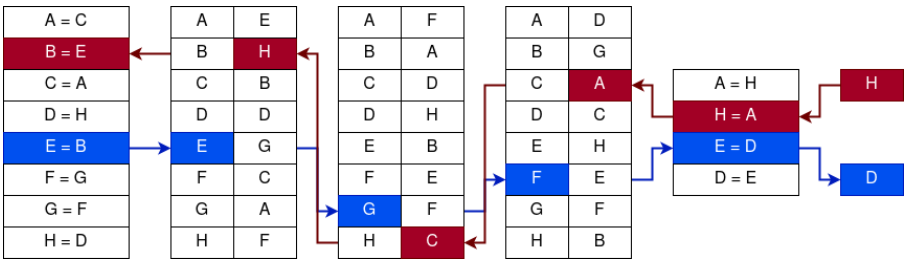

Рисунок 1.2 — Пример работы машина «Энигма» с коммутатором

2 Конструкторская часть

2.1 Схема алгоритма Энигма

На рисунке 2.1 представлен схема алгоритма машины «Энигма».

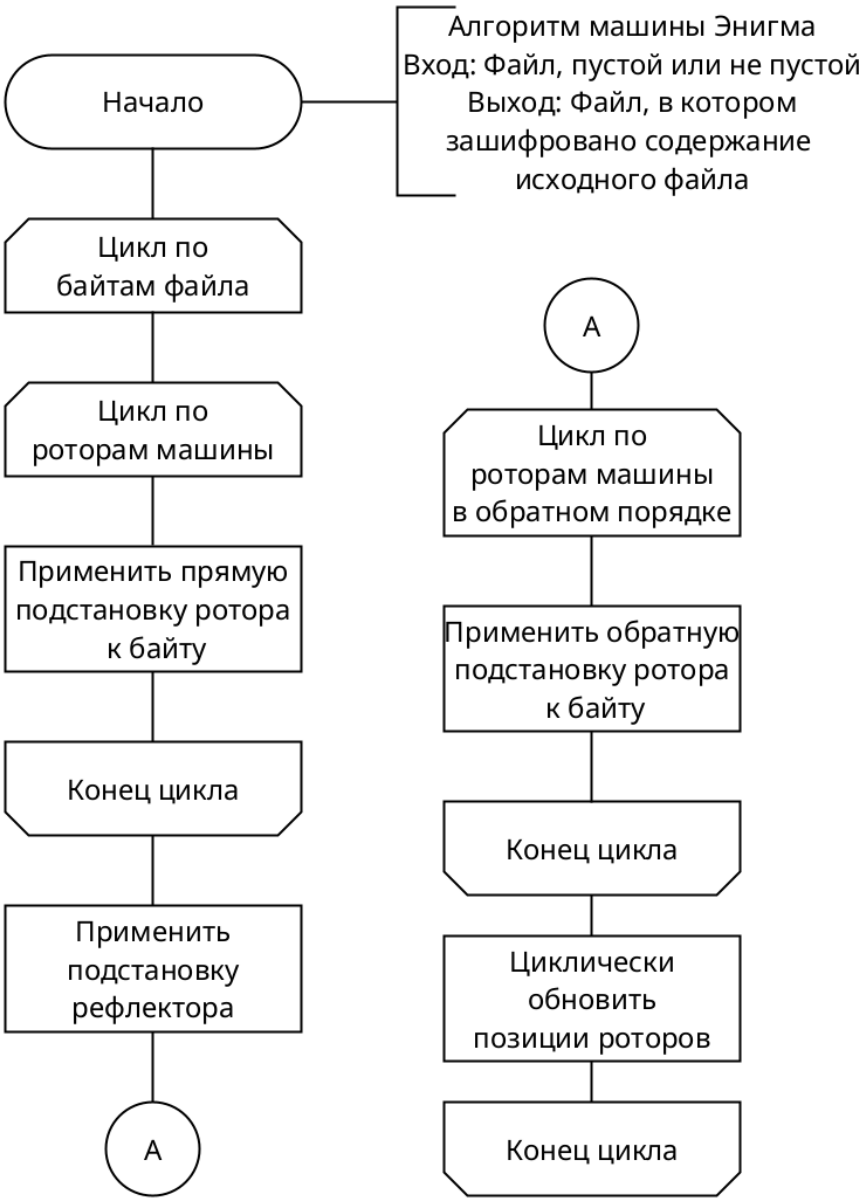

Рисунок 2.1 — Схема алгоритма машины «Энигма»

3 Технологическая часть

3.1 Средства реализации

В качестве языка программирования для программной реализации машины «Энигма» был выбран Go.

3.2 Код разработанной программы

На листингах 3.1- 3.5 представлена разработанная программа.

Листинг 3.1 — Код части машины Энигма

```
package enigma

type EnigmaPart interface {
  Apply(data byte) (byte, error)
  Update()
}

type MiddleEnigmaPart interface {
  EnigmaPart
  After(part EnigmaPart)
}
```

Листинг 3.2 — Код коммутатора машины Энигма

```
package enigma

import (
  "info-security/internal/utils/map_io"
  "math/rand"
  "strconv"
)

type Commutator struct {
  next EnigmaPart
  m    map[byte]byte
}

var _ MiddleEnigmaPart = (*Commutator)(nil)

func NewCommutator(next EnigmaPart, m map[byte]byte) *Commutator {
```

```
  return &Commutator{
    next: next,
    m:    m,
  }
}

func (c *Commutator) Apply(data byte) (byte, error) {
  if _, ok := c.m[data]; !ok {
    return 0, nil
  }
  return c.m[data], nil
}

func (c *Commutator) Update() {
  c.next.Update()
}

func (c *Commutator) After(part EnigmaPart) {
  c.next = part
}

func ReadCommutator(fName string) (*Commutator, error) {
  m, err := map_io.ReadMap(fName, func(s string) (byte, error) {
    b, err := strconv.ParseUint(s, 10, 8)
    if err != nil {
      return 0, err
    }
    return byte(b), nil
  })
  if err != nil {
    return nil, err
  }
  return NewCommutator(nil, m), nil
}

func DumpCommutator(c *Commutator, fName string) error {
  return map_io.DumpMap(c.m, fName)
}

func RandomCommutator() *Commutator {
```

```
  arr := make([]byte, 128)
  for i := 0; i < 128; i++ {
    arr[i] = byte(i) + 128
  }
  rand.Shuffle(128, func(i, j int) {
    arr[i], arr[j] = arr[j], arr[i]
  })
  m := make(map[byte]byte, 128)
  for i := 0; i < 128; i++ {
    m[byte(i)] = arr[i]
    m[arr[i]] = byte(i)
  }
  return NewCommutator(nil, m)
}
```

Листинг 3.3 — Код ротора машины Энигма

```
package enigma

import (
  "errors"
  "fmt"
  "info-security/internal/utils/bimap"
  "math/rand"
  "os"
  "strconv"
)

type EnigmaRotor struct {
  bmap      *bimap.BiMap[byte]
  next      EnigmaPart
  shift     byte
  shiftEdge byte
}

var RotorByteUnknownError error = errors.New("unknown byte")

var _ MiddleEnigmaPart = (*EnigmaRotor)(nil)

func NewEnigmaRotor(bmap *bimap.BiMap[byte], next EnigmaPart,
   shiftEdge byte) *EnigmaRotor {
```

```
  return &EnigmaRotor{
    bmap:      bmap,
    next:      next,
    shift:     0,
    shiftEdge: shiftEdge,
  }
}

func (r *EnigmaRotor) Apply(data byte) (byte, error) {
  data += r.shift
  d, ok := r.bmap.GetLeft(data)
  if !ok {
    return 0, RotorByteUnknownError
  }
  d2, err := r.next.Apply(d)
  if err != nil {
    return 0, err
  }
  d3, ok := r.bmap.GetRight(d2)
  if !ok {
    return 0, RotorByteUnknownError
  }
  return d3 - r.shift, nil
}

func (r *EnigmaRotor) After(part EnigmaPart) {
  r.next = part
}

func (r *EnigmaRotor) Update() {
  r.shift++
  if r.shift == r.shiftEdge {
    r.next.Update()
  }
}

func ReadEnigmaRotor(fName string) (*EnigmaRotor, error) {
  f, err := os.Open(fName)
  if err != nil {
    return nil, err
```

```
  }
  defer f.Close()

  var shiftEdge byte
  _, err = fmt.Fscanln(f, &shiftEdge)
  if err != nil {
    return nil, err
  }

  bmap, err := bimap.ReadBiMap[byte](f, func(s string) (byte, error)
      {
    b, err := strconv.ParseUint(s, 10, 8)
    if err != nil {
      return 0, err
    }
    return byte(b), nil
  })
  if err != nil {
    return nil, err
  }

  return NewEnigmaRotor(bmap, nil, shiftEdge), nil
}

func DumpEnigmaRotor(r *EnigmaRotor, fName string) error {
  f, err := os.OpenFile(fName, os.O_CREATE|os.O_WRONLY, 0644)
  if err != nil {
    return err
  }
  defer f.Close()
  _, err = fmt.Fprintf(f, "%d\n", r.shiftEdge)
  if err != nil {
    return err
  }
  return bimap.DumpBiMap(r.bmap, f)
}

func RandomEnigmaRotor() *EnigmaRotor {
  barr := make([]byte, 256)
  for i := 0; i < 256; i++ {
```

```
    barr[i] = byte(i)
  }
  rand.Shuffle(256, func(i, j int) {
    barr[i], barr[j] = barr[j], barr[i]
  })
  bmap := bimap.NewBiMap[byte]()
  for i := 0; i < 256; i++ {
    bmap.Add(byte(i), barr[i])
  }
  shiftEdge := byte(rand.Intn(256))
  return NewEnigmaRotor(bmap, nil, shiftEdge)
}
```

Листинг 3.4 — Код рефлектора машины Энигма

```
package enigma

import (
  "errors"
  "info-security/internal/utils/map_io"
  "math/rand"
  "strconv"
)

type EnigmaReflector struct {
  m map[byte]byte
}

var _ EnigmaPart = (*EnigmaReflector)(nil)

var ReflectorUnknownError error = errors.New("unknown byte")

func NewEnigmaReflector(m map[byte]byte) *EnigmaReflector {
  return &EnigmaReflector{
    m: m,
  }
}

func (r *EnigmaReflector) Apply(data byte) (byte, error) {
  if _, ok := r.m[data]; !ok {
    return 0, ReflectorUnknownError
```

```
  }
  return r.m[data], nil
}

func (r *EnigmaReflector) Update() {}

func ReadEnigmaReflector(fName string) (*EnigmaReflector, error) {
  m, err := map_io.ReadMap(fName, func(s string) (byte, error) {
    b, err := strconv.ParseUint(s, 10, 8)
    if err != nil {
      return 0, err
    }
    return byte(b), nil
  })
  if err != nil {
    return nil, err
  }
  return NewEnigmaReflector(m), nil
}

func DumpEnigmaReflector(r *EnigmaReflector, fName string) error {
  return map_io.DumpMap(r.m, fName)
}

func RandomEnigmaReflector() *EnigmaReflector {
  arr := make([]byte, 128)
  for i := 0; i < 128; i++ {
    arr[i] = byte(i) + 128
  }
  rand.Shuffle(128, func(i, j int) {
    arr[i], arr[j] = arr[j], arr[i]
  })
  m := make(map[byte]byte, 128)
  for i := 0; i < 128; i++ {
    m[byte(i)] = arr[i]
    m[arr[i]] = byte(i)
  }
  return NewEnigmaReflector(m)
}
```

Листинг 3.5 — Код машины Энигма

```
package enigma

import "fmt"

type EnigmaMachine struct {
  part EnigmaPart
}

func NewEnigmaMachine(part EnigmaPart) *EnigmaMachine {
  return &EnigmaMachine{
    part: part,
  }
}

func (m *EnigmaMachine) Apply(data []byte) ([]byte, error) {
  var result []byte
  for _, b := range data {
    r, err := m.part.Apply(b)
    if err != nil {
      return nil, fmt.Errorf("error applying part: %w", err)
    }
    result = append(result, r)
    m.part.Update()
  }
  return result, nil
}
```

3.3 Пример работы программы

Разработанная программа представляет из себя программу с интерфейсом командной строки. Программа считывает конфигурацию коммутатора, рефлектора и 3-х роторов из папки config, находящейся в той же директории, что и программа. Также она принимает два аргумента:

— **-input** – исходный файл для шифрования;

— **-output** – имя зашифрованного файла.

На рисунке 3.2 представлен пример шифрования архивного файла разработанным ПО.

```
./enigma --help
Usage of ./enigma:
  -input string
        input file
  -output string
        output file
```

Рисунок 3.1 — Описание аргументов программы

```
~/Projects/InfoSecurity main*
❯
./enigma -input docs.zip -output docs.enc.zip

~/Projects/InfoSecurity main*
❯
./enigma -input docs.enc.zip -output docs.dec.zip
```

Рисунок 3.2 — Пример шифрования архивного файла

На рисунках 3.3- 3.5 представлен архивный файл до шифрования, после и после расшифровки

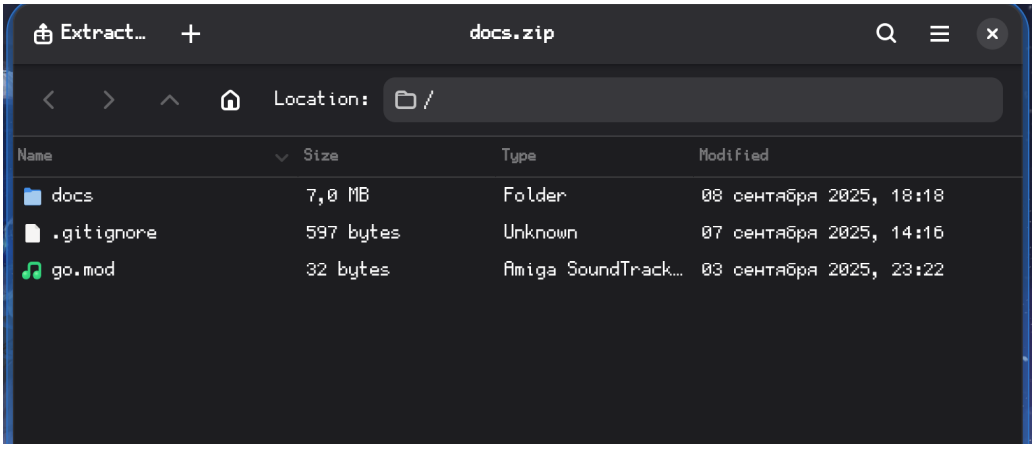

Рисунок 3.3 — Архивный файл до шифрования

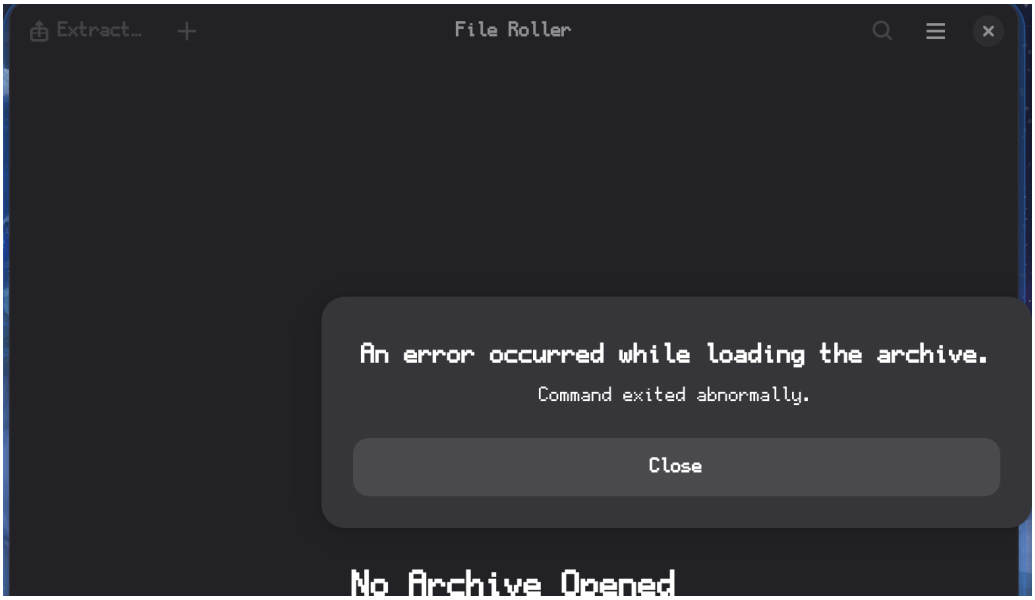

Рисунок 3.4 — Попытка открытия зашифрованного файла

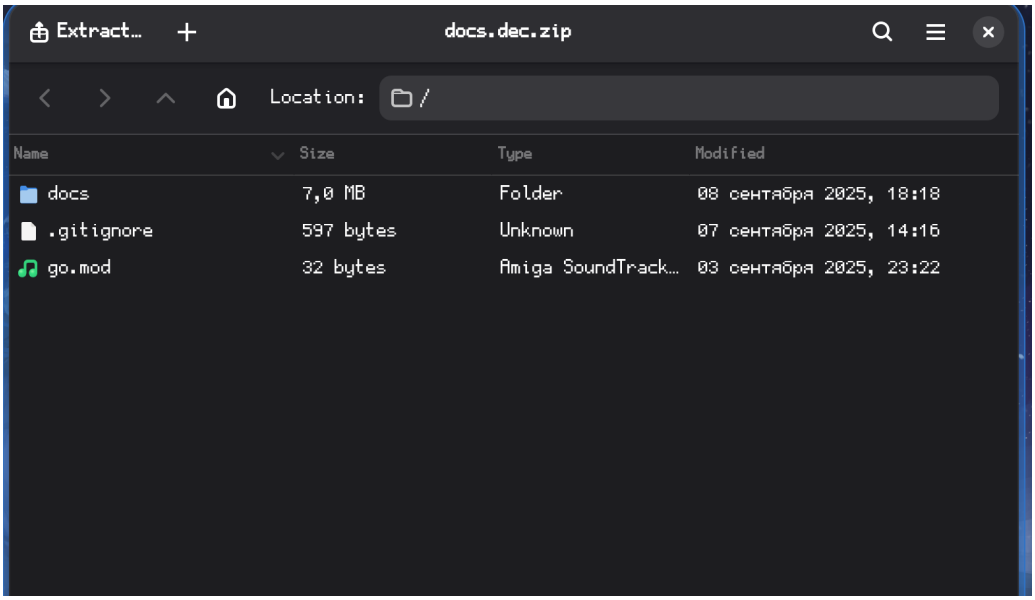

Рисунок 3.5 — Расшифрованный архивный файл

ЗАКЛЮЧЕНИЕ

В результате лабораторной работы был разработан электронный шифровальной машины «Энигма».

Были выполнены следующие задачи:

1) проанализирован принцип работы шифровальной машины «Энигма»;
2) составлена схема алгоритма шифровальной машины «Энигма»;
3) реализован описанный алгоритм шифрования;
4) проведено шифрование и расшифровка архивного файла;

Все поставленные цели и задачи были выполнены