Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика, искусственный интеллект и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ

по лабораторной работе № 1

по курсу «Защита Информации»

на тему: «Энигма»

Студент ИУ7-71Б (Группа) _____ (Подпись, дата) Корниенко К. Ю. (И. О. Фамилия)

Преподаватель _____ (Подпись, дата) Чиж И. С. (И. О. Фамилия)

2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель лабораторной работы — разработать программу шифровальной машины «Энигма» [1].

Задачи лабораторной работы:

— провести анализ устройства шифровальной машины «Энигма»;

— описать алгоритм шифрования «Энигмы»;

— реализовать описанный алгоритм;

— протестировать работу алгоритма на текстовых и бинарных файлах.

1 Аналитический раздел

Шифровальная машина «Энигма» состоит из трех основных частей:

1. роторы — диски, в классическом варианте обладающие 26 гранями, где каждая грань представляет собой символ латинского алфавита;

2. рефлектор — устройство, позволящие машине шифровать и расшифровывать текст по одному и тому же алгоритму;

3. коммутатор — набор парных шифров.

1.1 Алгоритм работы машины

На вход «Энигме» подается строка, которая разбивается на символы. Далее символ проходит через коммутационную панель, который меняет символ в соотвествии с настройкой. После прохождения панели, символ проходит черз три ротора и попадает на рефлектор. Каждый ротор изменяет исходный символ в соответствии со своим текущим положением. После работы рефлектора, символ отправляется обратно через роторы и оканчательно шифруется через коммутатор. Затем первый ротор совершает оборот, если он совершил полный оборот, то поворачивается следующий и так далее.

Благодаря наличию рефлектора, процессы шифрования и дешифровки идентичны.

2 Конструкторский раздел

2.1 Алгоритм работы Энигмы

На рисунке 2.1 приведена схема работы шифровальной машины Энигма.

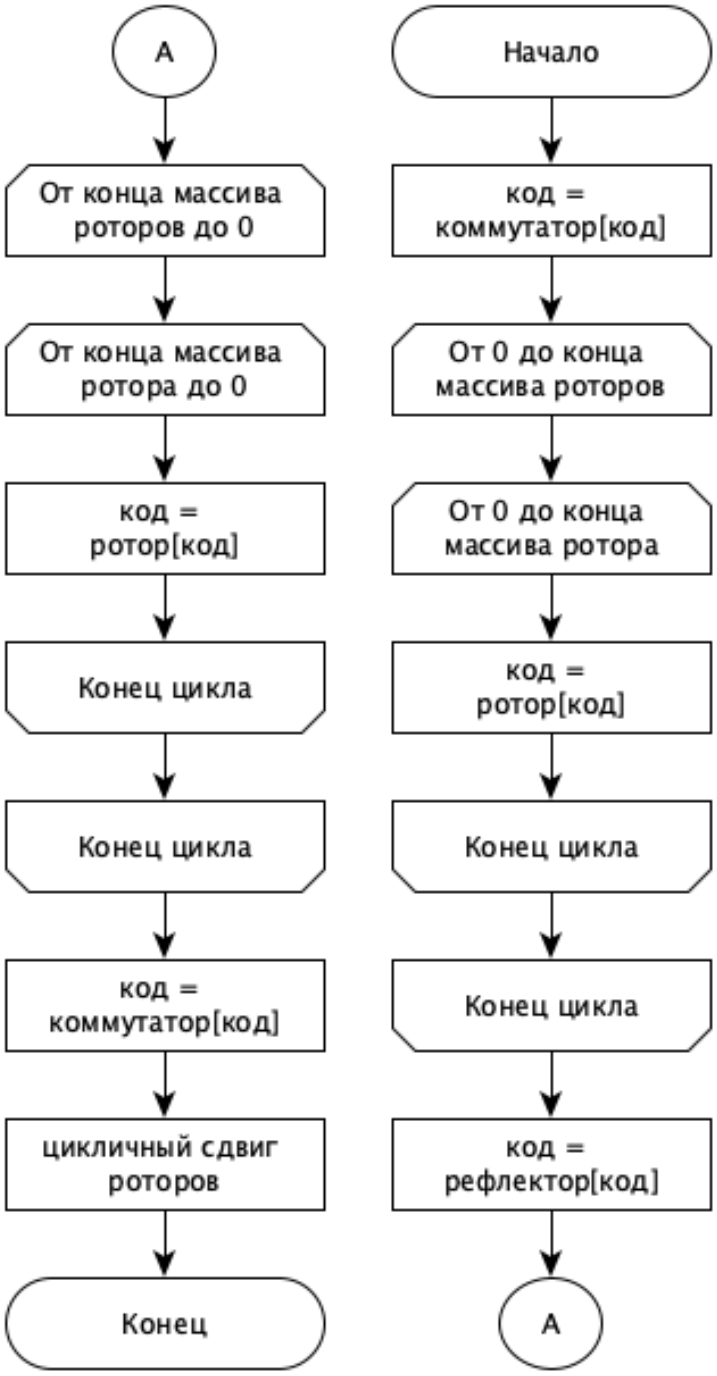

Рисунок 2.1 – Схема работы шифровальной машина Энигма

2.2 Модификация алгоритма

Классический алгоритм предусматривает шифрование произвольного текста, состоящего из печатных символов. Однако для обеспечения возможности шифровать произвольные бинарные файлы необходимо модифицировать классический алгоритм одним из двух способов.

1. Дополнить алгоритм преобразованием бинарных данных в текстовый формат (например, с помощью алгоритма base64).

   Однако в этом случае, алгоритмы шифрования и дешифровки будут разными, так как данное преобразование необходимо производить перед шифрованием и только для бинарных данных, и после дешифровки.

2. Расширить алфавит используемый в алгоритме до 256 символов.

   В этом случае идентичность процессов шифрования и дешифровки сохранится, а текстовые и бинарные данные будут обрабатываться одинаково.

В данной лабораторной работе применяется второй из описанных способов, как более простой и понятный в реализации.

3 Технологическая часть

3.1 Средства реализации

Для реализации ПО был выбран язык C++ [2]. Данный язык позволяет программировать в объектно-ориентированном стиле.

В качестве среды разработки была выбрана среда VS code [3]. В качестве средства сборки программы используется утилита Make.

3.2 Реализация алгоритма

На листинге 3.1 представлен класс представляющий шифровальную машину. Листинг 3.2 содержит класс, представляющий отдельный ротор. На листинге 3.3 представлен класс рефлектора.

Листинг 3.1 – Класс Энигмы

```
class Enigma final {
public:
    explicit Enigma(int seed) noexcept;
    ~Enigma() = default;

    void encrypt(std::istream& input, std::ostream& output);
private:
    void rotate();
    std::array<Rotor, 3> rotors;
    Reflector reflector;
};

Enigma::Enigma(int seed) noexcept :
   rotors{Rotor::buildRandom(seed), Rotor::buildRandom(seed +
   1), Rotor::buildRandom(seed + 2)} ,
   reflector(Reflector::buildRandom(seed + 3)) {}

void Enigma::encrypt(std::istream& input, std::ostream& output) {
    do {
        uint8_t symbol = input.get();
        if (input.eof()) break;

        symbol = rotors[0].forward(symbol);
        symbol = rotors[1].forward(symbol);
        symbol = rotors[2].forward(symbol);
```

```
        symbol = reflector.reflect(symbol);

        symbol = rotors[2].backward(symbol);
        symbol = rotors[1].backward(symbol);
        symbol = rotors[0].backward(symbol);

        output.put(symbol);
        rotate();
    } while (true);
}

void Enigma::rotate() {
    if (rotors[0].rotate())
        if (rotors[1].rotate())
            rotors[2].rotate();
}
```

Листинг 3.2 – Класс ротора

```
class Rotor final {
public:
    Rotor(const Rotor &) = delete;
    Rotor(Rotor &&) = default;
    Rotor &operator=(const Rotor &) = delete;
    Rotor &operator=(Rotor &&) = default;
    ~Rotor() = default;

    uint8_t forward(uint8_t input) const noexcept;
    uint8_t backward(uint8_t input) const noexcept;
    /* returns true on full rotation */
    bool rotate() noexcept;
    static Rotor buildRandom(int seed);
    static const int capacity = 256;
private:
    Rotor(std::array<uint8_t, 256> forward_pass,
       std::array<uint8_t, 256> backward_pass);
    int rotation = 0;
    std::array<uint8_t, 256> forward_pass;
    std::array<uint8_t, 256> backward_pass;
};

uint8_t Rotor::forward(uint8_t input) const noexcept {
```

```
    return forward_pass[input % capacity];
}

uint8_t Rotor::backward(uint8_t input) const noexcept {
    return backward_pass[input % capacity];
}

bool Rotor::rotate() noexcept {
    auto tmp = forward_pass[0];
    std::copy(forward_pass.begin() + 1, forward_pass.end(),
       forward_pass.begin());
    forward_pass[capacity - 1] = tmp;
    for (int pos = 0; pos < capacity; pos++)
        backward_pass[forward_pass[pos]] = pos;
    rotation = (rotation + 1) % capacity;
    return rotation == 0;
}

Rotor Rotor::buildRandom(int seed) {
    std::mt19937 random_generator(seed);
    std::array<uint8_t, 256> forward_pass;
    std::array<uint8_t, 256> backward_pass;

    for (int pos = 0; pos < capacity; pos++)
        forward_pass[pos] = pos;
    std::shuffle(forward_pass.begin(), forward_pass.end(),
       random_generator);
    for (int pos = 0; pos < capacity; pos++)
        backward_pass[forward_pass[pos]] = pos;
    return Rotor(forward_pass, backward_pass);
}
```

Листинг 3.3 – Класс рефлектора

```
class Reflector final {
public:
    Reflector(const Reflector &) = delete;
    Reflector(Reflector &&) = default;
    Reflector &operator=(const Reflector &) = delete;
    Reflector &operator=(Reflector &&) = default;
    ~Reflector() = default;

    uint8_t reflect(uint8_t input) const noexcept;
```

```
    static Reflector buildRandom(int seed) noexcept;
private:
    explicit Reflector(std::array<uint8_t, 256> wires) noexcept;
    std::array<uint8_t, 256> wires;
};

uint8_t Reflector::reflect(uint8_t input) const noexcept {
    return wires[input];
}

Reflector Reflector::buildRandom(int seed) noexcept {
    std::mt19937 random_generator(seed);
    std::array<uint8_t, 256> wires;
    wires.fill(0);
    int zero_pos = random_generator() % 256;
    wires[0] = zero_pos;
    int i = 0;
    while (1) {
        while (i < 256 && wires[i] != 0) i++;
        if (i >= 256) break;
        if (i == zero_pos) {
            i++;
            continue;
        }

        int pos = random_generator() % 256;
        while (wires[pos] != 0 || pos == zero_pos)
            pos = (pos + 1) % 256;
        wires[pos] = i;
        wires[i] = pos;
    }

    return Reflector(wires);
}
```

3.3 Тестирование ПО

Тестирование ПО проводится следующим образом. Для каждого из подготовленных входных файлов сначала запускается программа, которая создает зашифрованный файл. Затем полученные зашифрованные файлы снова подаются на вход программе. Полученные на втором шаге файлы сравниваются побайтово с исходными файлами и в случае отсутствия отличий тесты признаются успешно пройденными.

Вывод 16-ричного дампа файла производится с помощью стандартной утилиты `xxd`, побайтовое сравнение осуществляется с применением команды `diff`.

В таблице 3.1 приведены тесты для алгоритма шифрования Энигмы. Применена методология черного ящика. Тесты пройдены *успешно*.

Таблица 3.1 – Функциональные тесты для текстовых файлов

Файл	16-ричный дамп файла	
Входной файл 1	00000000: 7365 6372 6574 206d 00000008: 6573 7361 6765	secret m essage
Зашифрованный файл 1	00000000: 327b 33f9 248f 9d39 00000008: 655d 73df 265b	2{3.$..9 e]s.&.
Дешифрованный файл 1	00000000: 7365 6372 6574 206d 00000008: 6573 7361 6765	secret m essage
Входной файл 2	00000000: 3131 3131 3131 3131 00000008: 3131 3131 3131 3131 00000010: 3131 3131 3131 3131 00000018: 3131 3131 3131 310a	11111111 11111111 11111111 1111111.
Зашифрованный файл 2	00000000: d072 b537 61bd e940 00000008: 0e2b 9179 3144 1265 00000010: 31bc 879b cfb7 2268 00000018: 98e6 3535 67f1 2d0a	.r.7a..@ .+.y1D.e 1....."h ..55g.-.
Дешифрованный файл 2	00000000: 3131 3131 3131 3131 00000008: 3131 3131 3131 3131 00000010: 3131 3131 3131 3131 00000018: 3131 3131 3131 310a	11111111 11111111 11111111 1111111.

Таблица 3.2 – Функциональные тесты для бинарных файлов

Файл	16-ричный дамп начала файла
Входной файл 3	00000000: 8950 4e47 0d0a 1a0a .PNG.... 00000008: 0000 000d 4948 4452 ....IHDR 00000010: 0000 02c9 0000 02c7 .... 00000018: 0806 0000 007c 3fdf .....|?. ...
Зашифрованный файл 3	00000000: 113d 4afc 85b4 3982 .=J...9. 00000008: a449 000d d9d8 fb52 .I.....R 00000010: 9b78 a44c a200 027f .x.L.... 00000018: 27dd 70aa 0026 4cdf '.p..&L. ...
Дешифрованный файл 3	00000000: 8950 4e47 0d0a 1a0a .PNG.... 00000008: 0000 000d 4948 4452 ....IHDR 00000010: 0000 02c9 0000 02c7 .... 00000018: 0806 0000 007c 3fdf .....|?. ...
Входной файл 4	00000000: 7f45 4c46 0201 0100 .ELF.... 00000008: 0000 0000 0000 0000 .... 00000010: 0300 3e00 0100 0000 ..>..... 00000018: 4011 0000 0000 0000 @....... ...
Зашифрованный файл 4	00000000: 48cb 0cad d4d3 0100 H....... 00000008: a449 00f0 0000 40aa .I....@. 00000010: 0378 3a00 0100 2300 .x:...#. 00000018: 40c4 70aa 0000 d2d7 @.p..... ...
Дешифрованный файл 4	00000000: 7f45 4c46 0201 0100 .ELF.... 00000008: 0000 0000 0000 0000 .... 00000010: 0300 3e00 0100 0000 ..>..... 00000018: 4011 0000 0000 0000 @....... ...

ЗАКЛЮЧЕНИЕ

В данной лабораторной работе:

— проведен анализ работы шифровальной машины «Энигма»;

— описан алгоритм шифрования;

— реализован описанный алгоритм.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Prasad K.*, *Kumari M.* A review on mathematical strength and analysis of Enigma // arXiv preprint arXiv:2004.09982. — 2020.
2. *Josuttis N. M.* The C++ standard library: a tutorial and reference. — 2012.
3. *Code V. S.* Visual studio code // línea]. Available: https://code. visualstudio. com. — 2019.