Министерство науки и высшего образования РФ

Федеральное государственное автономное образовательное учреждение

высшего образования

«Московский политехнический университет»

(Московский политех)

Отчёт по курсу «Программирование криптографических алгоритмов»

Лабораторная работа 1.6 Поточные шифры



Выполнил:

Студент группы 221-352

Иванов В. В.

Проверил преподаватель: Бутакова Н. Г.

Москва 2024г.

**Аннотация**

* **Среда программирования**
  + Visual Studio Code
* **Язык программирования**
  + Python
* **Процедуры для запуска программы** 
  + Visual Studio Code (main.py)
* **Пословица-тест**
  + Тот, кто ложится на два стула, падает на ребра.
* **Текст для проверки работы (не меньше 1000 знаков (1430))**

Жизнь - это удивительное приключение, полное разнообразных событий и встреч. В каждом моменте мы находим что-то новое и уникальное. Стремление к росту и саморазвитию вдохновляет нас на поиск новых горизонтов. Важно помнить, что каждый шаг вперед приносит с собой уроки и опыт.

Разнообразие культур, языков и традиций делает наш мир удивительно богатым. Общение с людьми разных национальностей расширяет кругозор, позволяя нам понимать и уважать друг друга. Взаимное уважение и терпимость создают основу для гармоничного сосуществования.

Природа тоже играет важную роль в нашей жизни. Красота закатов, шум океана, пение птиц - все это напоминает нам о величии мира природы. Забота о окружающей среде становится неотъемлемой частью ответственного образа жизни.

Работа и творчество придают смысл нашим усилиям. Стремление к достижению целей мотивирует нас на новые начинания. Каждый проект, даже самый маленький, приносит удовлетворение и чувство выполненного долга.

Семья и друзья являются надежной опорой в нашей жизни. Обмен историями, веселые посиделки и поддержка в трудные моменты создают теплую атмосферу взаимопонимания и любви.

Таким образом, наша жизнь - это мозаика различных моментов, соединенных воедино. Важно ценить каждый момент и стремиться делать мир вокруг нас ярче и лучше. С любовью, терпением и целеустремленностью мы можем создавать свою уникальную историю, наполненную смыслом и радостью.

**15.А5 /1**

Символы исходного текста и гаммы представляются в виде двоичных кодов, а затем соответствующие разряды складываются по модулю

**Блок-схема программы**

**Код программы с комментариями**

import re

def majority(x1, x2, x3):

    return (x1 and x2) or (x1 and x3) or (x2 and x3)

def text2bin(text):

    return ''.join(format(ord(char), '016b') for char in text)

def bin2text(bin\_text):

    return ''.join(chr(int(bin\_text[i:i+16], 2)) for i in range(0, len(bin\_text), 16))

def A5CheckParameters(key):

    return bool(re.match(r'^[01]{64}$', key))

class LFSR:

    def \_\_init\_\_(self, length, xor\_bits, tact\_bits=[]):

        self.\_length = length

        self.\_xor\_bits = xor\_bits

        self.\_tact\_bits = tact\_bits

        self.\_lfsr = [0] \* length

        self.\_exit\_bit = 0

    @property

    def lfsr(self):

        return self.\_lfsr

    @lfsr.setter

    def lfsr(self, lfsr):

        self.\_lfsr = lfsr

    @property

    def tact\_bits(self):

        return [self.\_lfsr[self.\_length - bit - 1] for bit in self.\_tact\_bits]

    @property

    def exit\_bit(self):

        return self.\_exit\_bit

    def bit\_filling(self, bit):

        self.tact(bit\_filling=bit)

    def tact(self, shift=True, bit\_filling=0):

        res\_bit = 0

        for xor\_bit in self.\_xor\_bits:

            res\_bit ^= self.\_lfsr[self.\_length - xor\_bit - 1]

        self.\_exit\_bit = self.\_lfsr[0]

        if shift:

            self.\_lfsr.pop(0)

            self.\_lfsr.append(res\_bit ^ bit\_filling)

    def zeroize(self):

        self.\_lfsr = [0] \* self.\_length

def A51(open\_text, key, mode):

    encrypted\_text = ""

    R1 = LFSR(19, [13, 16, 17, 18], [8])

    R2 = LFSR(22, [20, 21], [10])

    R3 = LFSR(23, [7, 20, 21, 22], [10])

    bin\_text\_arr = []

    if mode == "encrypt":

        bin\_text\_arr = re.findall('.{1,114}', text2bin(open\_text))

    elif mode == "decrypt":

        bin\_text\_arr = re.findall('.{1,114}', open\_text)

    for i, bin\_text in enumerate(bin\_text\_arr):

        iter\_key = key + format(i, '022b')

        R1.zeroize()

        R2.zeroize()

        R3.zeroize()

        for bit in iter\_key:

            R1.bit\_filling(int(bit))

            R2.bit\_filling(int(bit))

            R3.bit\_filling(int(bit))

        for \_ in range(100):

            maj = majority(R1.tact\_bits[0], R3.tact\_bits[0], R2.tact\_bits[0])

            R1.tact(R1.tact\_bits[0] == maj)

            R2.tact(R2.tact\_bits[0] == maj)

            R3.tact(R3.tact\_bits[0] == maj)

        for bit in bin\_text:

            maj = majority(R1.tact\_bits[0], R3.tact\_bits[0], R2.tact\_bits[0])

            R1.tact(R1.tact\_bits[0] == maj)

            R2.tact(R2.tact\_bits[0] == maj)

            R3.tact(R3.tact\_bits[0] == maj)

            encrypted\_text += str(R1.exit\_bit ^ R2.exit\_bit ^ R3.exit\_bit ^ int(bit))

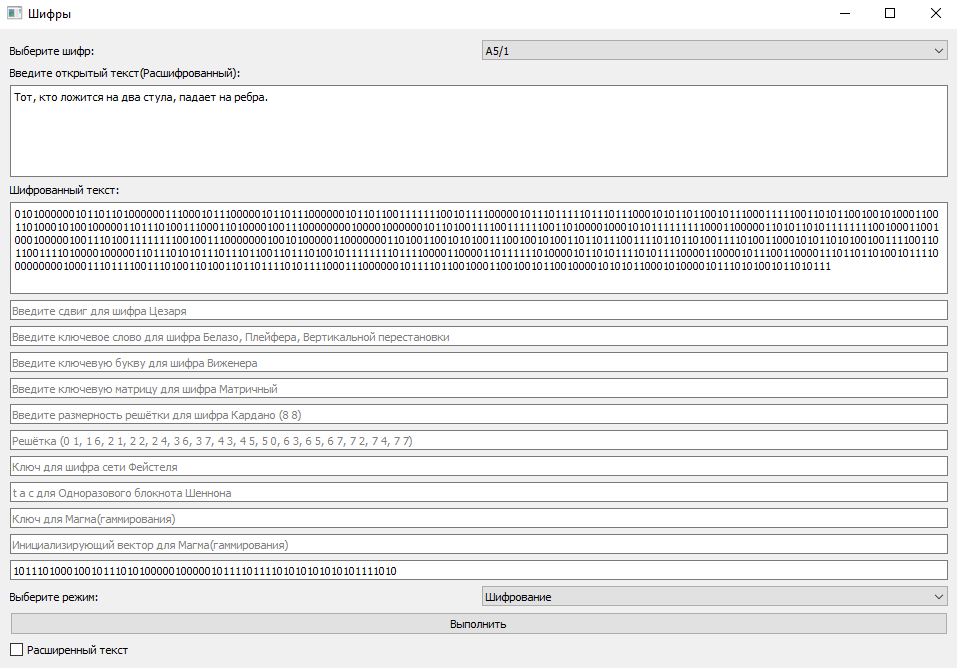
    if mode == "decrypt":

        encrypted\_text = bin2text(encrypted\_text).replace("тчк", ".").replace("зпт", ",").replace("тире", "-").replace('прбл', ' ').replace('двтч', ':').replace('тчсзп', ';').replace('отскб', '(').replace('зкскб', ')').replace('впрзн', '?').replace('восклзн', '!').replace('првст', '\n')

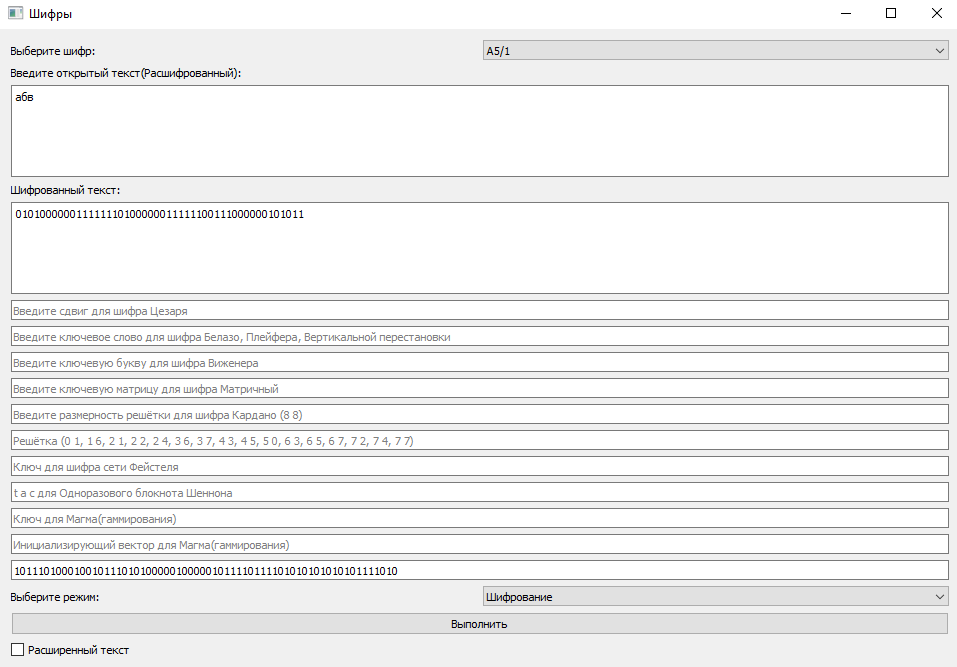
    return encrypted\_text

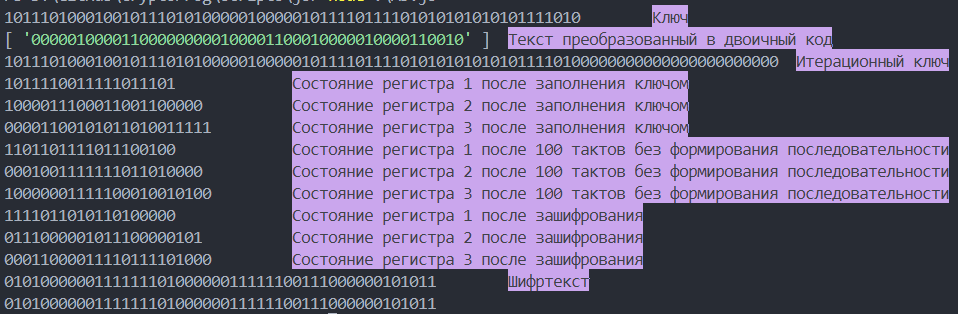
**Тестирование (скрин работы программы и фото карточки**

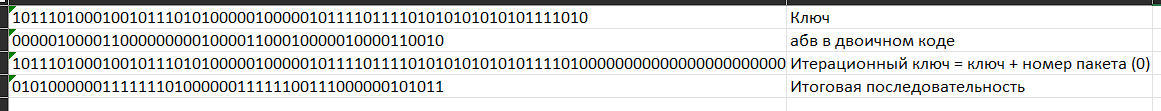
**ручного теста с указанием ключа)**



**Ручной тест**



****

****



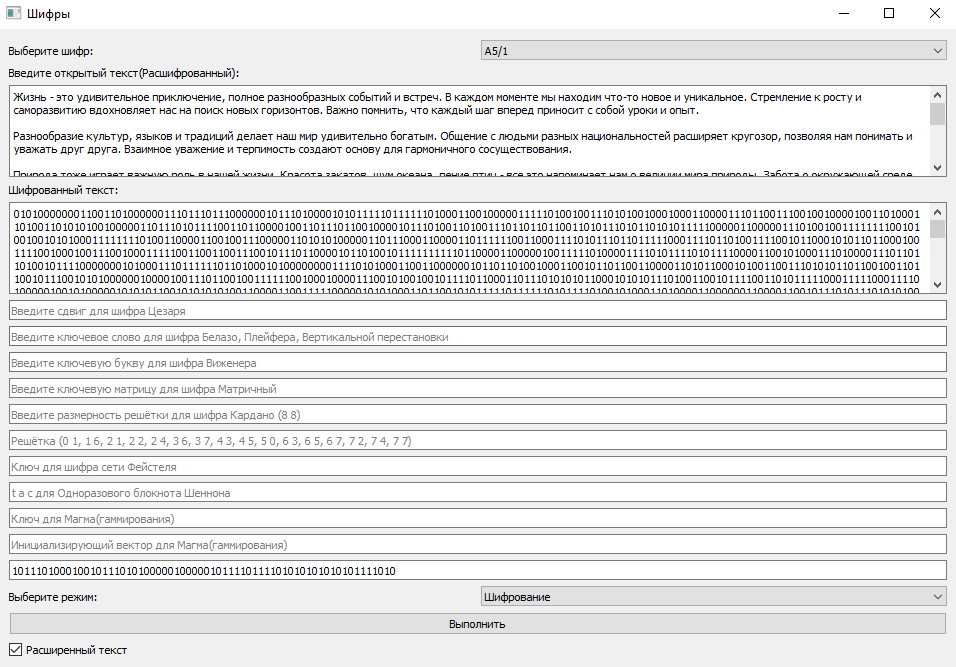




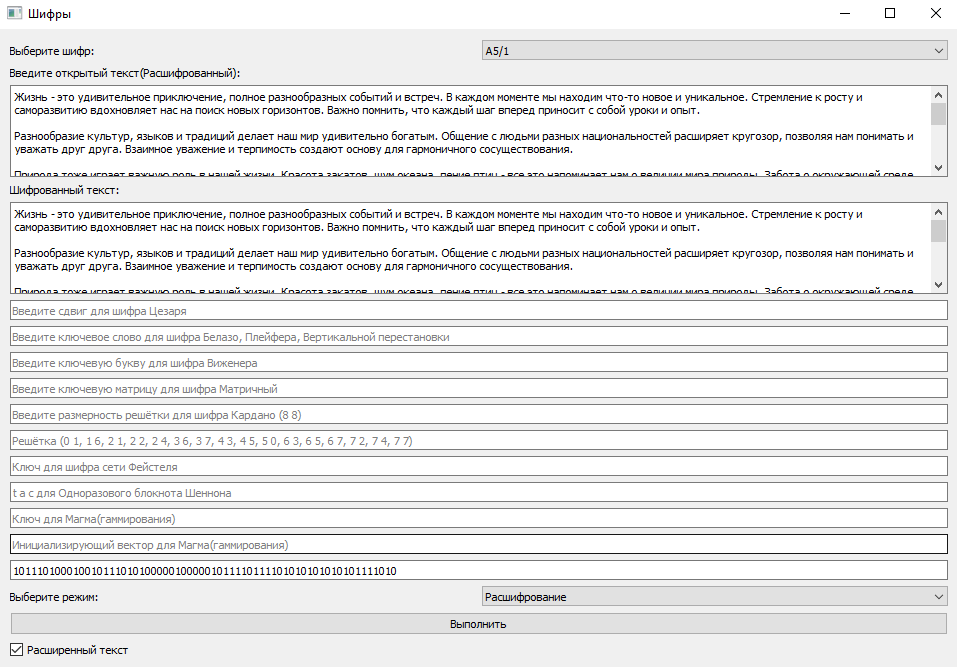
**Работа с текстом не менее 1000 знаков (шифрование и**

**расшифрование с указанием ключа)**

**Зашифрование**



**Расшифрование**

**16. А5 /2**

Символы исходного текста и гаммы представляются в виде двоичных кодов, а затем соответствующие разряды складываются по модулю 2.

**Блок-схема программы**

**Код программы с комментариями**

class LFSR {

    private \_length: number     *// length of lfsr*

    private \_xorBits: number[]  *// indexes of feedback polynomial (x^6 + x^5 + 1 equals [5, 4])*

    private \_tactBits: number[]    *// bit for clocking*

    private \_lfsr: number[]     *// linear feedback shift register*

    private \_exitBit: number    *// exit bit*

    constructor(*length*: number, *xorBits*: number[], *tactBits*: number[] = []) {

        this.\_length = *length*

        this.\_xorBits = *xorBits*

        this.\_tactBits = *tactBits*

        this.\_lfsr = new Array(*length*).fill(0)

        this.\_exitBit = 0

    }

    get lfsr() { return this.\_lfsr }

    set lfsr(*lfsr*: number[]) { this.\_lfsr = *lfsr*}

    get tactBits() {

        let tactBitValues: number[] = []

        for (const tactBit of this.\_tactBits) {

            tactBitValues.push(this.\_lfsr[this.\_length - tactBit - 1])

        }

        return tactBitValues

    }

    get exitBit() { return this.\_exitBit }

    bitFilling(*bit*: number) {

        this.tact(undefined, *bit*)

    }

    tact(*shift*: boolean = true, *bitFilling*: number = 0) {

        let resBit = 0

        for (const xorBit of this.\_xorBits) {

            resBit = this.\_lfsr[this.\_length - xorBit - 1] ^ resBit

        }

        this.\_exitBit = this.\_lfsr[0]

        if (*shift*) {

            this.\_lfsr.shift()

            this.\_lfsr.push(resBit ^ *bitFilling*)

        }

    }

    zeroize() {

        this.\_lfsr = new Array(this.\_length).fill(0)

    }

}

function majority(*x1*: number, *x2*: number, *x3*: number): number {

    return *x1* && *x2* || *x1* && *x3* || *x2* && *x3*

}

function text2bin(*text*: string) {

    return *text*.split("").map(*char* => ("0000000000000000" + *char*.charCodeAt(0).toString(2)).slice(-16)).join("")

}

function bin2text(*binText*: string) {

    let text = ""

    for (let i = 0; i < *binText*.length; i += 16) {

        const bin = *binText*.slice(i, i + 16);

        text += String.fromCharCode(parseInt(bin, 2))

    }

    return text

}

export function A5CheckParameters(*key*: string): boolean{

    return /[0,1]{64}/.test(*key*) && *key*.length == 64

}

export function A52(*openText*: string, *key*: string, *mode*: string): string {

    let encryptedText: string = "" *// Шифртекст*

    const R1 = new LFSR(19, [13, 16, 17, 18])

    const R2 = new LFSR(22, [20, 21])

    const R3 = new LFSR(23, [7, 20, 21, 22])

    const R4 = new LFSR(17, [11, 16], [3, 7, 10])

    let binTextArr: string[] = []

    if (*mode* == "encrypt")

        binTextArr = text2bin(*openText*).match(/.{1,114}/g) ?? []

    else if (*mode* == "decrypt")

        binTextArr = *openText*.match(/.{1,114}/g) ?? []

    for (let i = 0; i < binTextArr.length; i++) {

*///////////////////// Initialization /////////////////////*

*// Iteration key*

        const iterKey = *key* + ("0000000000000000000000" + i.toString(2)).slice(-22)

*// Zeroing registers*

        R1.zeroize()

        R2.zeroize()

        R3.zeroize()

        R4.zeroize()

*// Initializing filling of registers*

        for (const bit of iterKey) {

            R1.bitFilling(Number(bit))

            R2.bitFilling(Number(bit))

            R3.bitFilling(Number(bit))

            R4.bitFilling(Number(bit))

        }

*// Set R4(3) = 1, R4(7) = 1, R4(10) = 1*

        const R4lfsr = R4.lfsr

        R4lfsr[3] = 1

        R4lfsr[7] = 1

        R4lfsr[10] = 1

        R4.lfsr = R4lfsr

*// First 99 tacts*

        for (let i = 0; i < 100; i++) {

            const maj = majority(R4.tactBits[0], R4.tactBits[1], R4.tactBits[2])

            R1.tact(R1.tactBits[3] == maj)

            R2.tact(R2.tactBits[1] == maj)

            R3.tact(R3.tactBits[2] == maj)

            R4.tact()

        }

*///////////////////// Encryption /////////////////////*

        for (const binText of binTextArr[i]) {

            const maj = majority(R4.tactBits[0], R4.tactBits[1], R4.tactBits[2])

            const majr1 = majority(R1.lfsr[12], R1.lfsr[14], R1.lfsr[15])

            const majr2 = majority(R1.lfsr[9], R1.lfsr[13], R1.lfsr[16])

            const majr3 = majority(R1.lfsr[13], R1.lfsr[16], R1.lfsr[18])

            R1.tact(R4.tactBits[2] == maj)

            R2.tact(R4.tactBits[0] == maj)

            R3.tact(R4.tactBits[1] == maj)

            R4.tact()

            encryptedText += R1.exitBit ^ R2.exitBit ^ R3.exitBit ^ majr1 ^ majr2 ^ majr3 ^ Number(binText)

        }

    }

*// Перевод символов из их текстовых значений в символьные*

    if (*mode* == "decrypt")

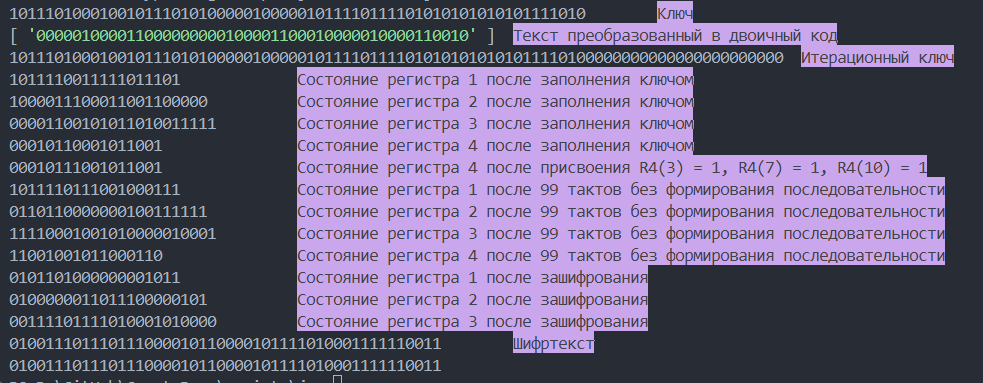
        encryptedText = bin2text(encryptedText).replaceAll("тчк", ".").replaceAll("зпт", ",").replaceAll("тире", "-").replaceAll('прбл', ' ').replaceAll('двтч', ':').replaceAll('тчсзп', ';').replaceAll('отскб', '(').replaceAll('зкскб', ')').replaceAll('впрзн', '?').replaceAll('восклзн', '!').replaceAll('првст', '\n');

    return encryptedText *// Возврат шифртекста*

}

**Тестирование**

**Ручной тест**

****









**Работа с текстом не менее 1000 знаков (шифрование и**

**расшифрование с указанием ключа)**

**Зашифрование**

**Расшифрование**