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 8. Асимметричные шифры



Выполнил:

Студент группы 221-352

Герасенков И. И.

Проверил преподаватель: Бутакова Н. Г.

Москва 2024г.

**Аннотация**

* **Среда программирования**
  + Visual Studio Code
* **Язык программирования**
  + JavaScript (TypeScript)
* **Процедуры для запуска программы** 
  + Перейти по ссылке https://cryptoprog.netlify.app/
* **Пословица-тест**
  + Заяц в кустах – это еще не готовое мясо.
* **Текст для проверки работы (не меньше 1000 знаков (1950))**
  + А судьи кто? - За древностию лет

К свободной жизни их вражда непримирима,

Сужденья черпают из забытых газет

Времён Очаковских и покоренья Крыма;

Всегда готовые к журьбе,

Поют все песнь одну и ту же,

Не замечая об себе:

Что старее, то хуже.

Где? укажите нам, отечества отцы,

Которых мы должны принять за образцы?

Не эти ли, грабительством богаты?

Защиту от суда в друзьях нашли, в родстве,

Великолепные соорудя палаты,

Где разливаются в пирах и мотовстве,

И где не воскресят клиенты-иностранцы

Прошедшего житья подлейшие черты.

Да и кому в Москве не зажимали рты

Обеды, ужины и танцы?

Не тот ли, вы к кому меня ещё с пелен,

Для замыслов каких-то непонятных,

Дитёй возили на поклон?

Тот Нестор негодяев знатных,

Толпою окружённый слуг;

Усердствуя, они в часы вина и драки

И честь, и жизнь его не раз спасали: вдруг

На них он выменял борзые три собаки!!!

Или вон тот ещё, который для затей

На крепостной балет согнал на многих фурах

От матерей, отцов отторженных детей?!

Сам погружён умом в Зефирах и в Амурах,

Заставил всю Москву дивиться их красе!

Но должников не согласил к отсрочке:

Амуры и Зефиры все

Распроданы поодиночке!!!

Вот те, которые дожили до седин!

Вот уважать кого должны мы на безлюдьи!

Вот наши строгие ценители и судьи!

Теперь пускай из нас один,

Из молодых людей, найдётся - враг исканий,

Не требуя ни мест, ни повышенья в чин,

В науки он вперит ум, алчущий познаний;

Или в душе его сам бог возбудит жар

К искусствам творческим, высоким и прекрасным, -

Они тотчас: разбой! пожар!

И прослывёт у них мечтателем! опасным!! -

Мундир! один мундир! он в прежнем их быту

Когда-то укрывал, расшитый и красивый,

Их слабодушие, рассудка нищету;

И нам за ними в путь счастливый!

И в жёнах, дочерях - к мундиру та же страсть!

Я сам к нему давно ль от нежности отрёкся?!

Теперь уж в это мне ребячество не впасть;

Но кто б тогда за всеми не повлекся?

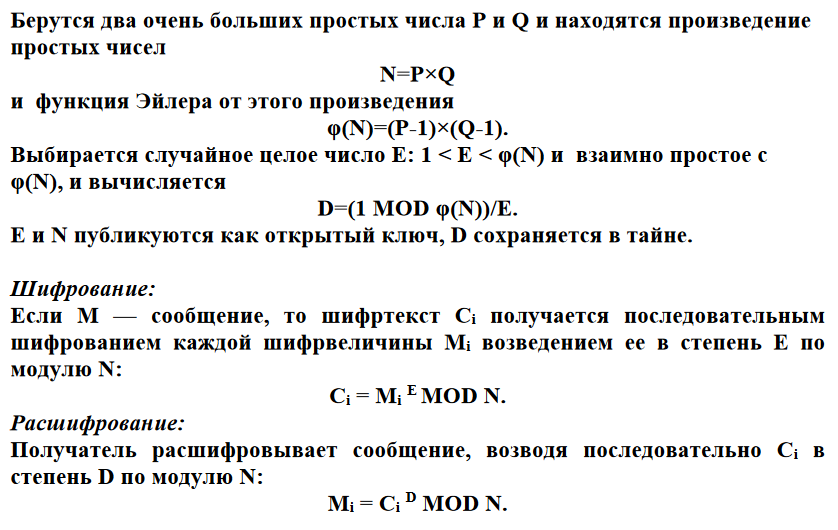
Когда из гвардии, иные от двора

Сюда на время приезжали, -

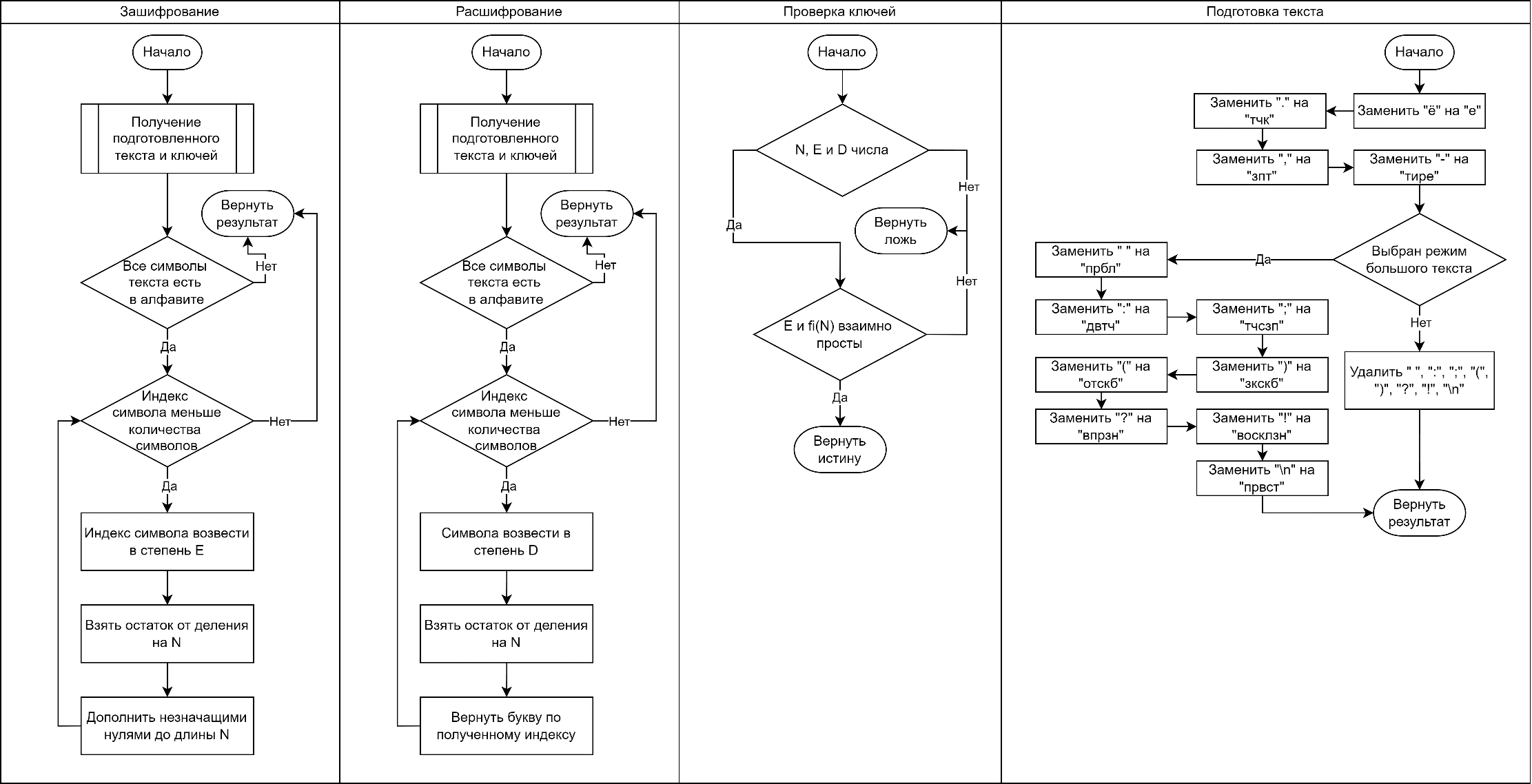
Кричали женщины: ура!

И в воздух чепчики бросали!

**21.RSA**



**Блок-схема программы**

****

**Код программы с комментариями**

function gcd(a: number, b: number): number { // Greatest common divisor

    while (b) {

        [a, b] = [b, a % b]

    }

    return a

}

function fi(n: number): number {

    let num = 0

    for (let i = 1; i < n; i++) {

        if (gcd(i, n) == 1)

            num++

    }

    return num

}

function isPrime(num: number): boolean {

    if (num <= 1) return false

    for (let i = 2, sqrt = Math.sqrt(num); i <= sqrt; i++)

        if (num % i === 0) return false

    return true

}

function comparison(comp: number[]): number { // comp = [x, y, z] xa = ymod(z)

    for (let y = 0; y < comp[2]; y++) {

        if (((comp[0] \* y) % comp[2]) == (comp[1] % comp[2]))

            return y

    }

    return 0

}

function findCoprime(n: number): number[] { // Returns the list of n coprimes

    let coprimes: number[] = []

    for (let i = 2; i < n; i++)

        if (gcd(i, n) == 1) coprimes.push(i)

    return coprimes

}

function setAutocompleteE(es: number[]) {

    const rsaEAutocomplete = document.querySelector("#rsa-e-autocomplete") as HTMLElement

    rsaEAutocomplete.innerHTML = ""

    for (const e of es) {

        const listItem = document.createElement("li")

        listItem.classList.add("rsa-e-autocomplete-item")

        listItem.textContent = String(e)

        rsaEAutocomplete.appendChild(listItem)

    }

}

function setD(fin: number, e: number): string {

    const rsaDInput = document.querySelector(".rsa-d-input") as HTMLInputElement

    const d = comparison([e, 1, fin])

    rsaDInput.value = String(d)

    if (d == e) return "D equals E, encryption is useless"

    return ""

}

export function RSACheckParameters(p: number, q: number, e: number, d: number): string {

    if (!(p && q)) return "p or q is NaN" // p, q, e or d is NaN

    if (!(isPrime(p) && isPrime(q))) return "p or q is not prime" // p, q, e or d is NaN

    if (p == q) return "p == q" // p, q, e or d is NaN

    if (p \* q < 32) return "p \* q < 32"

    if (!e) {

        setAutocompleteE(findCoprime(fi(p \* q)))

        return "set the parameter e"

    }

    return setD(fi(p \* q), e)

    // if ((p && q && e))

    // if (d != comparison([e, 1, fi(p \* q)])) return false // wrong d

    // if (gcd(e, fi(p \* q)) != 1) return "e and fi(n) not mutually prime" // e and fi(n) not mutually prime

    // return "" // Everything is ok

}

export function RSAEncrypt(openText: string, p: number, q: number, e: number, alphabet: string[]): string {

    for (const letter of openText) {

        if (!alphabet.includes(letter)) return "Введёный текст содержит запрещённые символы" // Буквы текста не содержатся в алфавите

    }

    const n = p \* q

    let encryptedText: string = "" // Шифртекст

    for (const letter of openText) {

        console.log(BigInt(alphabet.indexOf(letter) + 1), BigInt(e), BigInt(n));

        encryptedText += ("0".repeat(String(n).length) + (BigInt(alphabet.indexOf(letter) + 1) \*\* BigInt(e)) % BigInt(n)).slice(-String(n).length)

    }

    return encryptedText // Возврат шифртекста

}

export function RSADecrypt(encryptedText: string, p: number, q: number, e: number, d: number, alphabet: string[]): string {

    let decriptedText: string = "" // Шифртекст

    const n = p \* q

    // from "1234567890" to [12, 34, 56, 78, 90]

    const lenLetter = String(n).length

    const encryptedTextArr = (encryptedText.match(new RegExp(`.{1,${lenLetter}}`, "g")) ?? []).map(num => Number(num))

    for (const letter of encryptedTextArr) {

        decriptedText += alphabet[Number(((BigInt(letter) \*\* BigInt(d)) % BigInt(n)) - BigInt(1))]

    }

    // Перевод символов из их текстовых значений в символьные

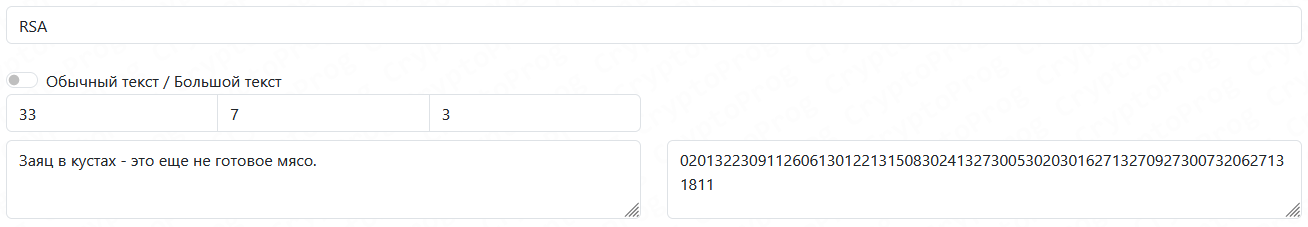
    decriptedText = decriptedText.replaceAll("тчк", ".").replaceAll("зпт", ",").replaceAll("тире", "-").replaceAll('прбл', ' ').replaceAll('двтч', ':').replaceAll('тчсзп', ';').replaceAll('отскб', '(').replaceAll('зкскб', ')').replaceAll('впрзн', '?').replaceAll('восклзн', '!').replaceAll('првст', '\n')

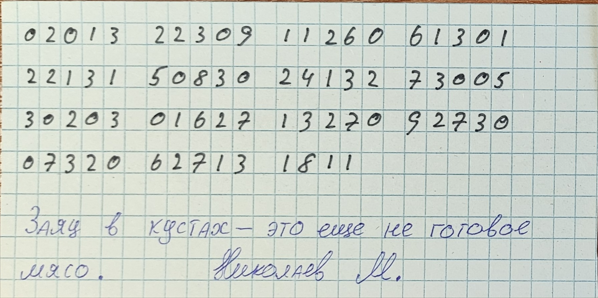
    return decriptedText // Возврат шифртекста

}

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

**ручного теста с указанием ключа)**

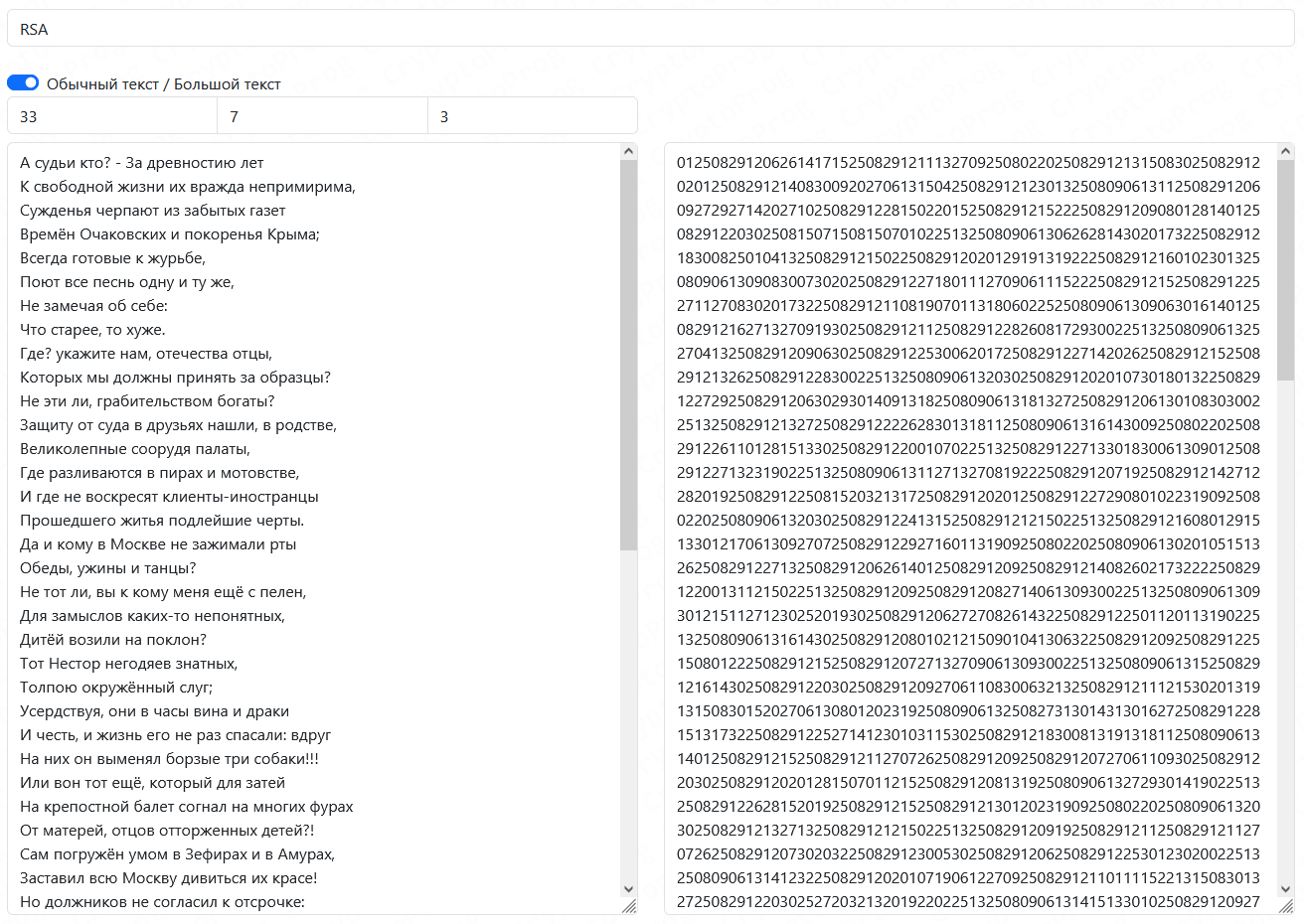
****

****

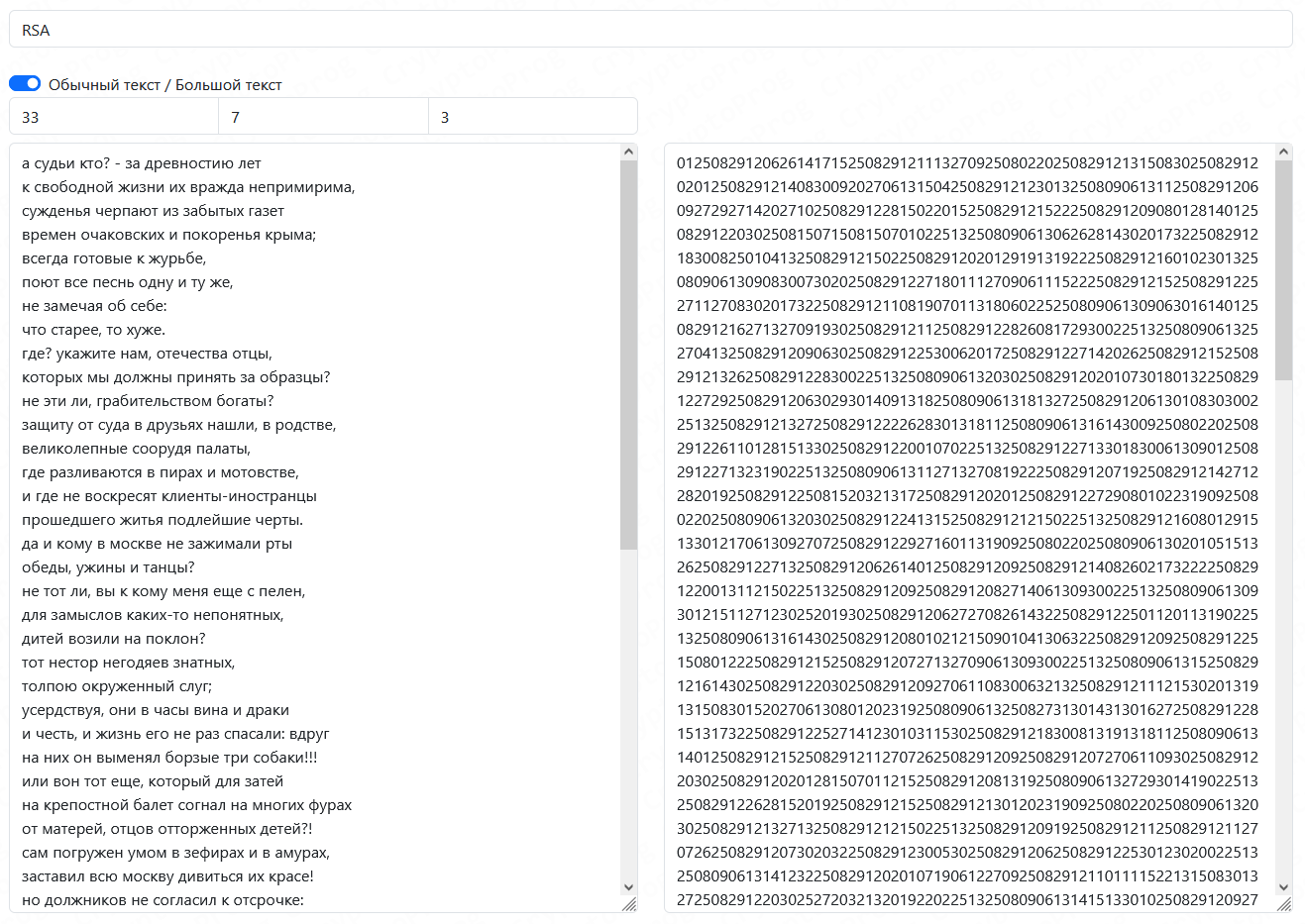
**Работа с текстом не менее 1000 знаков (шифрование и**

**расшифрование с указанием ключа)**

**Зашифрование**

****

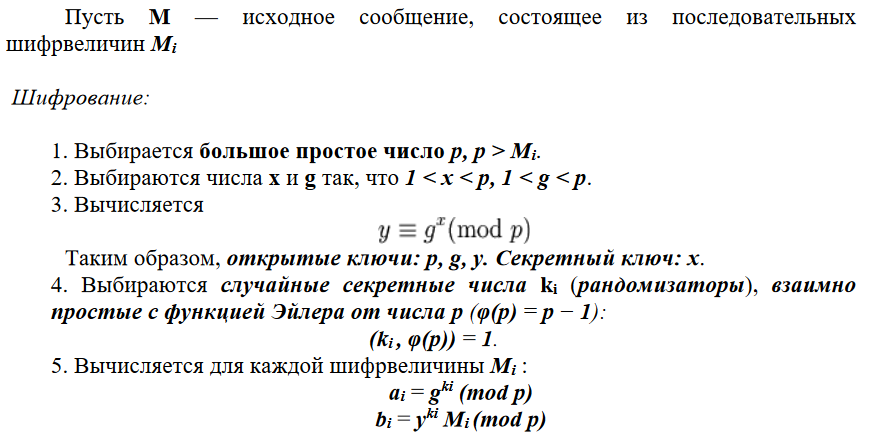
**Расшифрование**

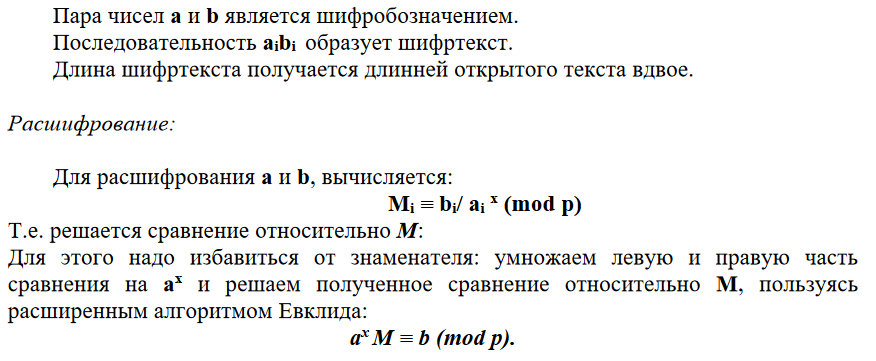
****

**Исполняемый файл**

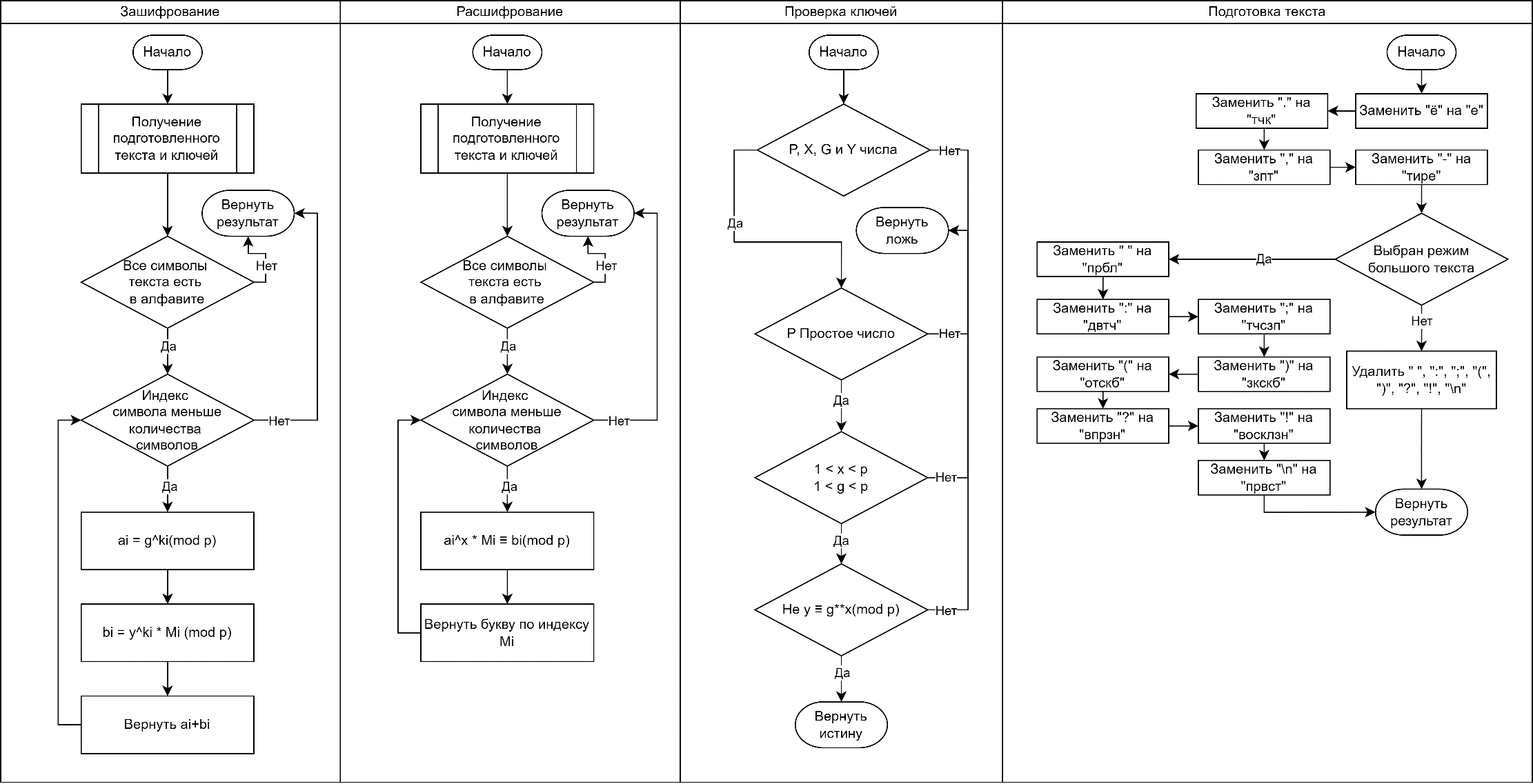
Не существует, есть ссылка <https://cryptoprog.netlify.app/>

**22.Elgamal**





**Блок-схема программы**

****

**Код программы с комментариями**

function gcd(a: number, b: number): number { // Greatest common divisor

    while (b) [a, b] = [b, a % b]

    return a

}

function findCoprime(n: number): number[] { // Returns the list of n coprimes

    let coprimes: number[] = []

    for (let i = 2; i < n; i++)

        if (gcd(i, n) == 1) coprimes.push(i)

    return coprimes

}

function fi(n: number): number {

    let num = 0

    for (let i = 1; i < n; i++) {

        if (gcd(i, n) == 1)

            num++

    }

    return num

}

function comparison(comp: number[]): number { // comp = [x, y, z] xa = ymod(z)

    for (let y = 0; y < comp[2]; y++) {

        if (((comp[0] \* y) % comp[2]) == (comp[1] % comp[2]))

            return y

    }

    return 0

}

function isPrime(num: number): boolean {

    if (num <= 1) return false

    for (let i = 2, sqrt = Math.sqrt(num); i <= sqrt; i++)

        if (num % i === 0) return false

    return true

}

function setY(p: number, x: number, g: number) {

    const elgamalYInput = document.querySelector(".elgamal-y-input") as HTMLInputElement

    elgamalYInput.value = String((BigInt(g) \*\* BigInt(x)) % BigInt(p))

}

export function elgamalCheckParameters(p: number, x: number, g: number) {

    if (!(p && x && g)) return "p, x or g is NaN" // p, x, g or y is NaN

    if (p < 32) return "p < 32" // p is not a prime

    if (!isPrime(p)) return "p is not a prime" // p is not a prime

    if (!((1 < x && x < p) && (1 < g && g < p))) return "Not 1 < x < p, 1 < g < p" // Not 1 < x < p, 1 < g < p

    setY(p, x, g)

    return "" // Everything is ok

}

export function elgamalEncrypt(openText: string, p: number, x: number, g: number, y: number, alphabet: string[]): string {

    for (const letter of openText) {

        if (!alphabet.includes(letter)) return "Введёный текст содержит запрещённые символы" // Буквы текста не содержатся в алфавите

    }

    const keys = findCoprime(fi(p))

    let encryptedText: string = "" // Шифртекст

    for (const letter of openText) {

        const ki = keys[Math.floor(Math.random() \* keys.length)]

        const ai = ("0".repeat(String(p).length) + ((BigInt(g) \*\* BigInt(ki)) % BigInt(p))).slice(-String(p).length)

        const bi = ("0".repeat(String(p).length) + ((BigInt(y) \*\* BigInt(ki) \* BigInt(alphabet.indexOf(letter))) % BigInt(p))).slice(-String(p).length)

        encryptedText += ai + bi

    }

    return encryptedText // Возврат шифртекста

}

export function elgamalDecrypt(encryptedText: string, p: number, x: number, g: number, y: number, alphabet: string[]): string {

    const keys = findCoprime(fi(p))

    let decriptedText: string = "" // Шифртекст

    // from "12345678" to [[12, 34], [56, 78]]

    const lenLetter = String(p).length

    const encryptedTextArr = (encryptedText.match(new RegExp(`.{1,${lenLetter \* 2}}`, "g")) ?? []).map(char => [Number(char.slice(0, char.length / 2)), Number(char.slice(char.length / 2))])

    for (const [ai, bi] of encryptedTextArr) {

        decriptedText += alphabet[comparison([Number((BigInt(ai) \*\* BigInt(x)) % BigInt(p)), bi, p])]

    }

    // Перевод символов из их текстовых значений в символьные

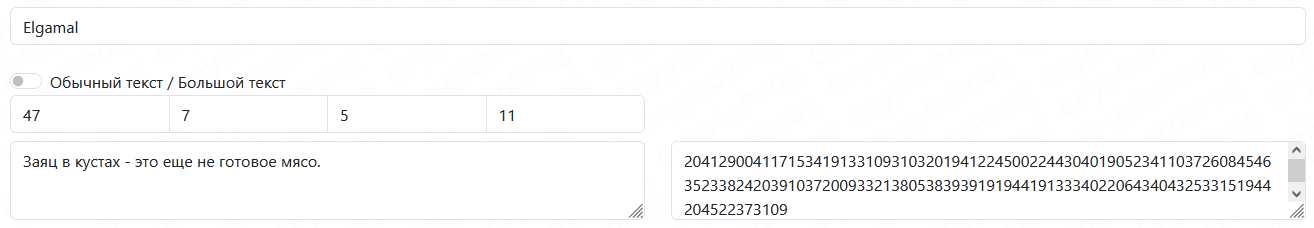
    decriptedText = decriptedText.replaceAll("тчк", ".").replaceAll("зпт", ",").replaceAll("тире", "-").replaceAll('прбл', ' ').replaceAll('двтч', ':').replaceAll('тчсзп', ';').replaceAll('отскб', '(').replaceAll('зкскб', ')').replaceAll('впрзн', '?').replaceAll('восклзн', '!').replaceAll('првст', '\n')

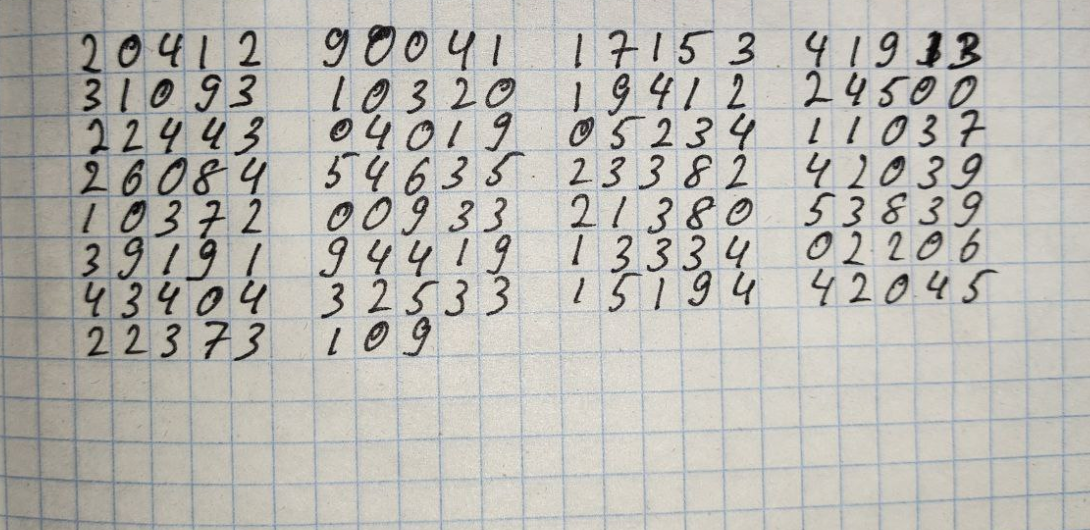
    return decriptedText // Возврат шифртекста

}

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

**ручного теста с указанием ключа)**

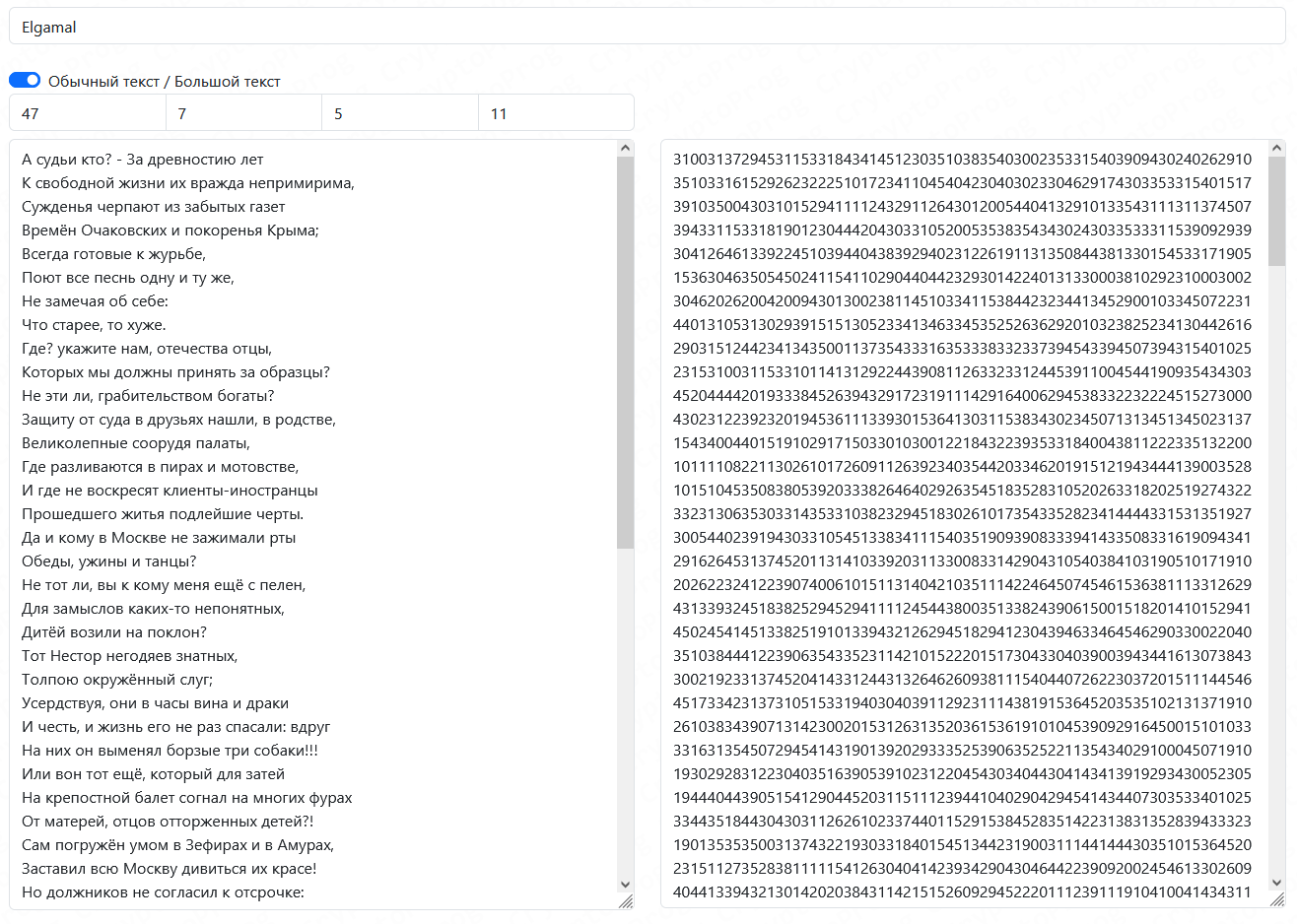
****

****

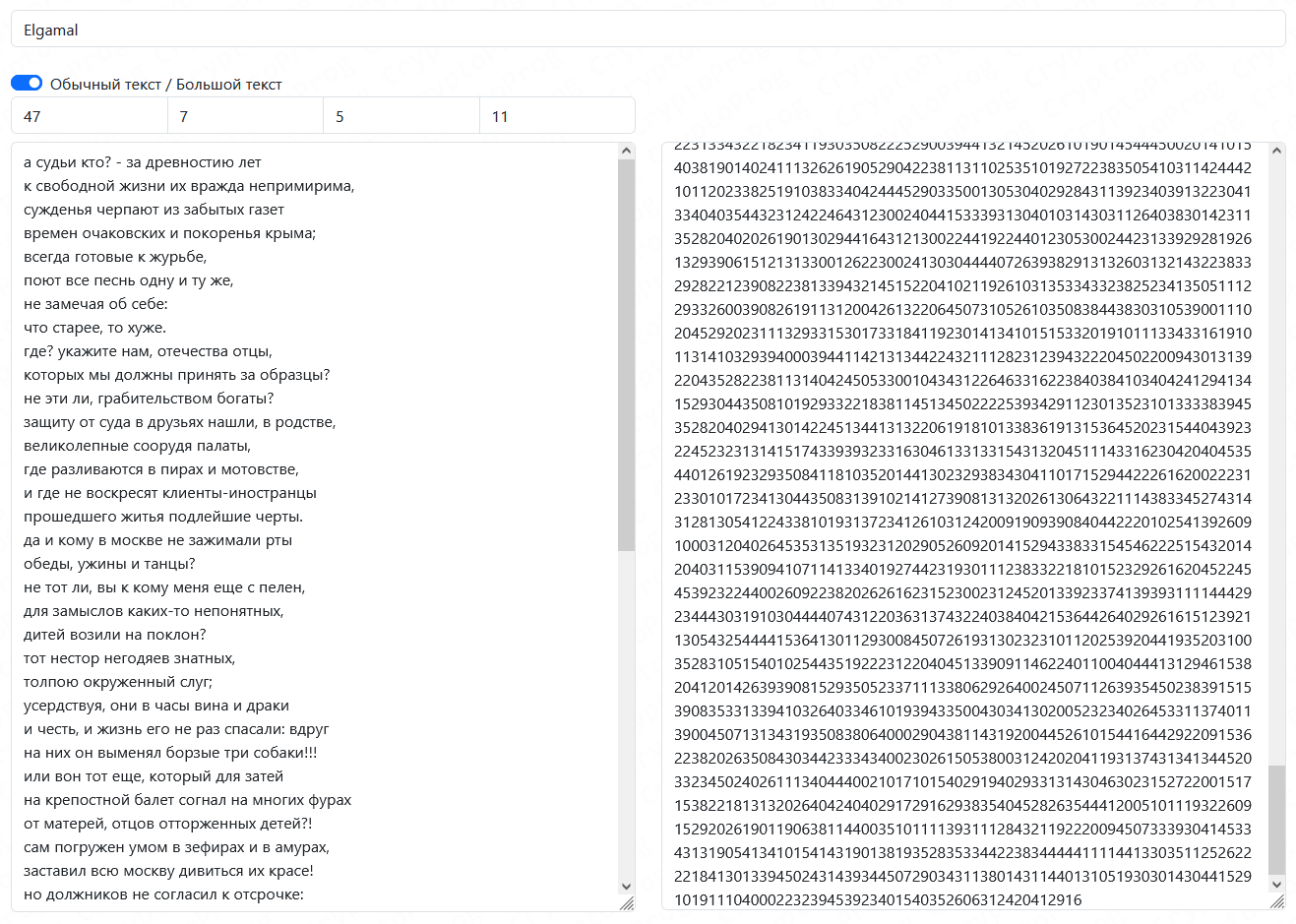
**Работа с текстом не менее 1000 знаков (шифрование и**

**расшифрование с указанием ключа)**

**Зашифрование**

****

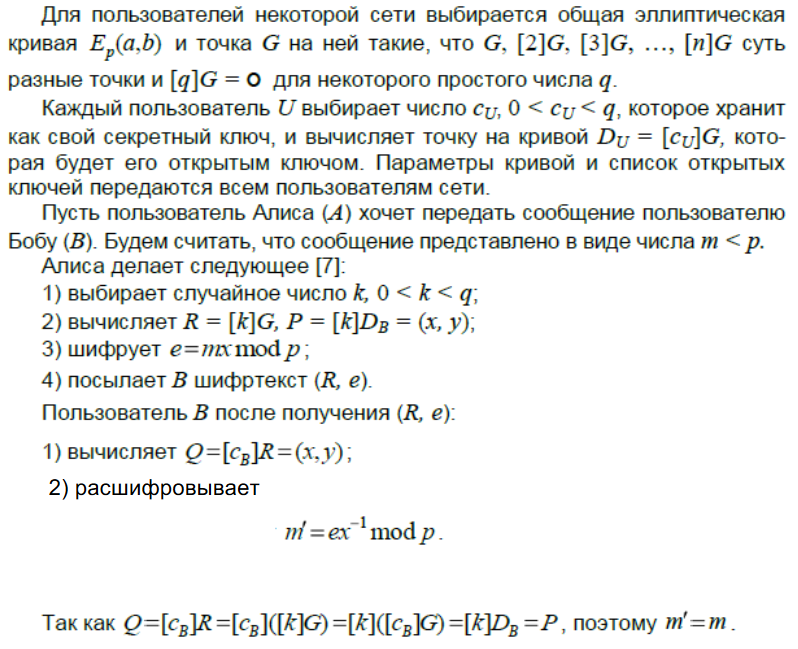
**Расшифрование**

****

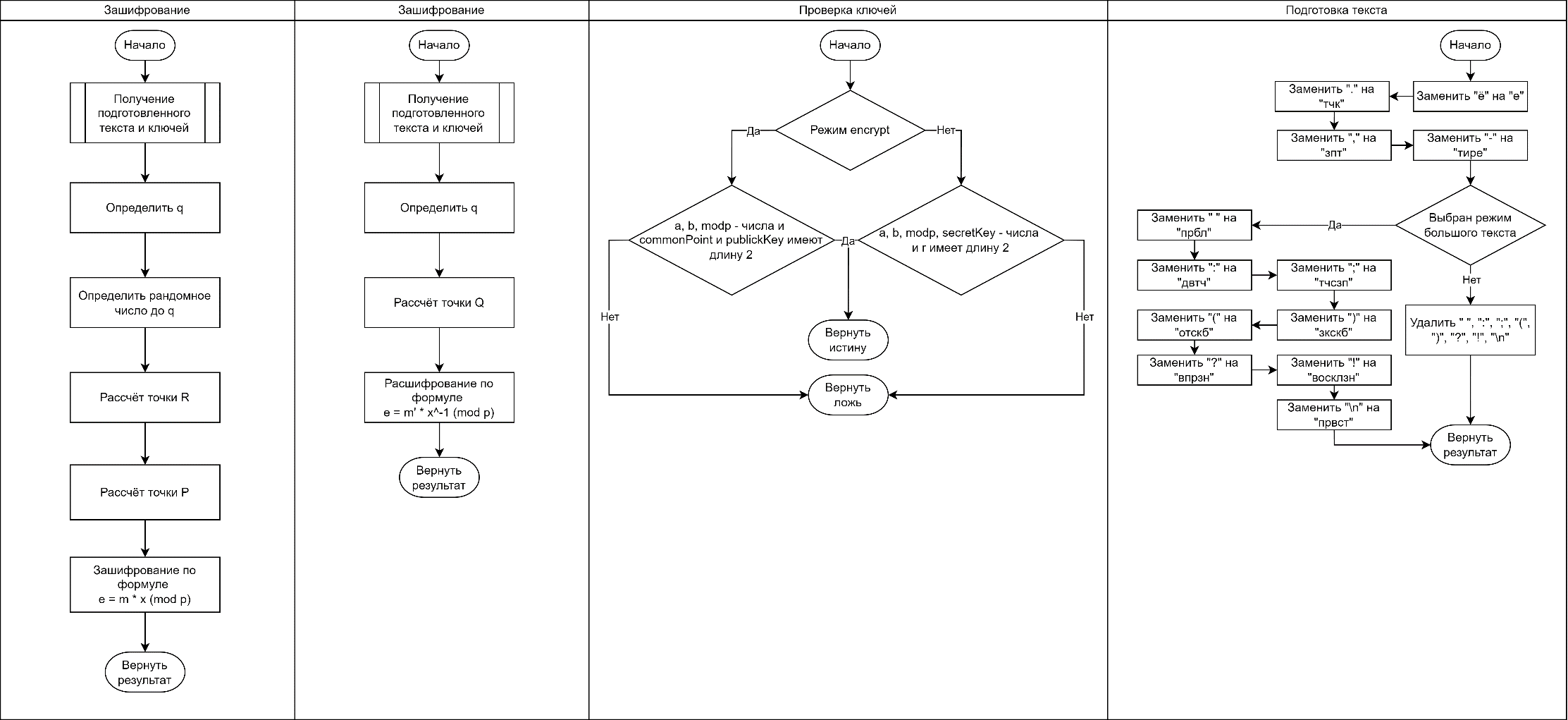
**Исполняемый файл**

Не существует, есть ссылка <https://cryptoprog.netlify.app/>

**23.ECC – С ИСПОЛЬЗОВАНИЕМ АБСЦИССЫ ТОЧКИ**



**Блок-схема программы**

****

**Код программы с комментариями**

class EllipticCurves {

    //                                                      (y^2 = x^3 + ax + b) mod p

    private \_a: number              // coefficient a in the →→→→→→→→→→→→→↑    ↑      ↑

    private \_b: number              // coefficient b in the →→→→→→→→→→→→→→→→→→↑      ↑

    private \_p: number              // coefficient p in the →→→→→→→→→→→→→→→→→→→→→→→→→↑

    private \_commonPoint: number[]  // common point

    private \_secretKey: number      // secret key

    private \_randomNum: number      // random number

    private \_publicKey: number[]    // public key

    private \_numMessage: string     // number message

    private \_q: number              // point group order

    private \_h: number              // cofactor

    constructor(

        a: number,

        b: number,

        p: number,

        commonPoint: number[] = [0, 0],

        publicKey: number[] = [0, 0],

        secretKey: number = 0,

        randomNum: number = 0,

        numMessage: string = "",

        q: number = 0,

        h: number = 0,

    ) {

        this.\_a = a

        this.\_b = b

        this.\_p = p

        this.\_commonPoint = commonPoint

        this.\_publicKey = publicKey     // Du = [Cu]G

        this.\_secretKey = secretKey     // 0 < Cu < q

        this.\_randomNum = randomNum     // 0 < k < q

        this.\_numMessage = numMessage

        this.\_q = q

        this.\_h = h

    }

    set commonPoint(commonPoint: number[]) { this.\_commonPoint = commonPoint }

    get commonPoint() { return this.\_commonPoint }

    set secretKey(secretKey: number) { this.\_secretKey = secretKey }

    get secretKey() { return this.\_secretKey }

    set randomNum(randomNum: number) { this.\_randomNum = randomNum }

    get randomNum() { return this.\_randomNum }

    set publicKey(publicKey: number[]) { this.\_publicKey = publicKey }

    get publicKey() { return this.\_publicKey }

    set numMessage(numMessage: string) { this.\_numMessage = numMessage }

    get numMessage() { return this.\_numMessage }

    get q() { return this.\_q }

    gcd(a: number, b: number): number { // Greatest common divisor

        while (b) {

            [a, b] = [b, a % b]

        }

        return a

    }

    fi(n: number): number {

        let num = 0

        for (let i = 1; i < n; i++) {

            if (this.gcd(i, n) == 1)

                num++

        }

        return num

    }

    defineQ() {

        // array of prime numbers up to n

        function primeSieve(n: number): number[] {

            let a = Array(Math.ceil(n = n / 2)),

                t = (Math.sqrt(4 + 8 \* n) - 2) / 4,

                u = 0,

                r: number[] = [];

            for (let i = 1; i <= t; i++) {

                u = (n - i) / (1 + 2 \* i);

                for (let j = i; j <= u; j++) a[i + j + 2 \* i \* j] = true;

            }

            for (let i = 0; i <= n; i++) !a[i] && r.push(i \* 2 + 1);

            return r;

        }

        const sqxArr: number[] = Array.from(Array(this.\_p), (\_, index) => (index \*\* 2) % this.\_p); // array of x\*\*2 (0 < x < p)

        let q = 0

        // point count

        for (let i = 0; i < this.\_p; i++) {

            const sqy = (i \*\* 3 + this.\_a \* i + this.\_b) % this.\_p

            const num = sqxArr.find(num => num === sqy)

            if (num)

                q += 2

            else if (num == 0)

                q += 1

        }

        q += 1 // infinity point

        const primeArr = primeSieve(q)

        for (const primeNum of primeArr.reverse()) {

            if (q % primeNum == 0) {

                this.\_q = primeNum

                this.\_h = q / primeNum

                break

            }

        }

    }

    formMessage(message: string) {

        const alphabet: string[] = ["а", "б", "в", "г", "д", "е", "ж", "з", "и", "й", "к", "л", "м", "н", "о", "п", "р", "с", "т", "у", "ф", "х", "ц", "ч", "ш", "щ", "ъ", "ы", "ь", "э", "ю", "я"]

        let numMessage = ""

        for (const letter of message) {

            numMessage += String(alphabet.indexOf(letter)).padStart(String(this.\_p).length, "0")

        }

        this.\_numMessage = numMessage

    }

    pointSum(point1: number[], point2: number[]): number[] {

        const numerator = (((point2[1] - point1[1]) % this.\_p) + this.\_p) % this.\_p

        const denominator = (((point2[0] - point1[0]) % this.\_p) + this.\_p) % this.\_p

        let lambda = 0

        if ((numerator / denominator) % 1 == 0) lambda = numerator / denominator

        else lambda = (numerator \* denominator \*\* (this.fi(this.\_p) - 1)) % this.\_p

        const x = (((lambda \*\* 2 - point1[0] - point2[0]) % this.\_p) + this.\_p) % this.\_p

        const y = (((lambda \* (point1[0] - x) - point1[1]) % this.\_p) + this.\_p) % this.\_p

        return [x, y]

    }

    pointDouble(point: number[]): number[] {

        const numerator = (((3 \* point[0] \*\* 2 + this.\_a) % this.\_p) + this.\_p) % this.\_p

        const denominator = (((2 \* point[1]) % this.\_p) + this.\_p) % this.\_p

        let lambda = 0

        if ((numerator / denominator) % 1 == 0) lambda = numerator / denominator

        else lambda = (numerator \* denominator \*\* (this.fi(this.\_p) - 1)) % this.\_p

        const x = (((lambda \*\* 2 - 2 \* point[0]) % this.\_p) + this.\_p) % this.\_p

        const y = (((lambda \* (point[0] - x) - point[1]) % this.\_p) + this.\_p) % this.\_p

        return [x, y]

    }

    calcPoint(prefix: number, point: number[]): number[] {

        const actions: string = prefix.toString(2).slice(1)

        let resPoint: number[] = point

        for (const action of actions) {

            resPoint = this.pointDouble(resPoint)

            if (action == "1") resPoint = this.pointSum(resPoint, point)

        }

        return resPoint

    }

    encrypt(x: number): string {

        let encryptedText: string = "" // Шифртекст

        for (const num of this.\_numMessage.match(new RegExp(`.{1,${String(this.\_p).length}}`, "g")) ?? []) {

            encryptedText += String((((parseInt(num) \* x) % this.\_p) + this.\_p) % this.\_p).padStart(String(this.\_p).length, "0")

        }

        return encryptedText // Возврат шифртекста

    }

    decrypt(e: string, x: number): string {

        let decriptedText: string = "" // Шифртекст

        for (const num of e.match(new RegExp(`.{1,${String(this.\_p).length}}`, "g")) ?? []) {

            decriptedText += String((((parseInt(num) \* (x \*\* (this.fi(this.\_p) - 1))) % this.\_p) + this.\_p) % this.\_p).padStart(String(this.\_p).length, "0")

        }

        return decriptedText // Возврат шифртекста

    }

}

export function eccCheckParameters(

    a: number,

    b: number,

    modp: number,

    commonPoint: number[],

    publicKey: number[],

    secretKey: number,

    r: number[],

    mode: string

): boolean {

    if (mode == "encrypt" && a && b && modp && commonPoint.length == 2 && publicKey.length == 2) return true

    else if (mode == "decrypt" && a && b && modp && secretKey && r.length == 2) return true

    return false

}

export function eccEncrypt(

    a: number,

    b: number,

    modp: number,

    commonPoint: number[],

    publicKey: number[] = [-1, -1],

    secretKey: number = -1,

    openText: string,

    randomNum: number = -1) {

    const eccEnc = new EllipticCurves(a, b, modp, commonPoint)

    eccEnc.defineQ()

    if (randomNum == -1) eccEnc.randomNum = Math.floor(Math.random() \* (eccEnc.q))

    else eccEnc.randomNum = randomNum

    const alphabetLetter: string[] = ["а", "б", "в", "г", "д", "е", "ж", "з", "и", "й", "к", "л", "м", "н", "о", "п", "р", "с", "т", "у", "ф", "х", "ц", "ч", "ш", "щ", "ъ", "ы", "ь", "э", "ю", "я"]

    const alphabetNumber: string[] = ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"]

    let flagLetter: boolean = true

    let flagNums: boolean = true

    for (const letter of openText) {

        if (!alphabetLetter.includes(letter)) {

            flagLetter = false

            break

        }

    }

    for (const num of openText) {

        if (!alphabetNumber.includes(num)) {

            flagNums = false

            break

        }

    }

    if (!(flagLetter || flagNums)) return "Введёный текст содержит запрещённые символы"

    if (flagLetter) eccEnc.formMessage(openText)

    else eccEnc.numMessage = openText

    if (publicKey[0] == -1 && publicKey[1] == -1) {

        const d = eccEnc.calcPoint(secretKey, eccEnc.commonPoint)

        eccEnc.publicKey = d

    }

    else eccEnc.publicKey = publicKey

    const r = eccEnc.calcPoint(eccEnc.randomNum, eccEnc.commonPoint)

    const p = eccEnc.calcPoint(eccEnc.randomNum, eccEnc.publicKey)

    return `((${r[0]}, ${r[1]}), ${eccEnc.encrypt(p[0])})`

}

export function eccDecrypt(

    a: number,

    b: number,

    modp: number,

    r: number[],

    secretKey: number,

    encryptedText: string) {

    const eccDec = new EllipticCurves(a, b, modp, undefined, undefined, secretKey)

    eccDec.defineQ()

    const alphabetLetter: string[] = ["а", "б", "в", "г", "д", "е", "ж", "з", "и", "й", "к", "л", "м", "н", "о", "п", "р", "с", "т", "у", "ф", "х", "ц", "ч", "ш", "щ", "ъ", "ы", "ь", "э", "ю", "я"]

    const alphabetNumber: string[] = ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"]

    let flagNums: boolean = true

    for (const num of encryptedText) {

        if (!alphabetNumber.includes(num)) {

            flagNums = false

            break

        }

    }

    if (!flagNums) return "Введёный текст содержит запрещённые символы"

    else eccDec.numMessage = encryptedText

    const q = eccDec.calcPoint(eccDec.secretKey, r)

    const decriptedText = eccDec.decrypt(encryptedText, q[0])

    const decriptedTextArr = decriptedText.match(new RegExp(`.{1,${String(modp).length}}`, "g")) ?? []

    let preparedDecriptedText = ""

    for (const num of decriptedTextArr)

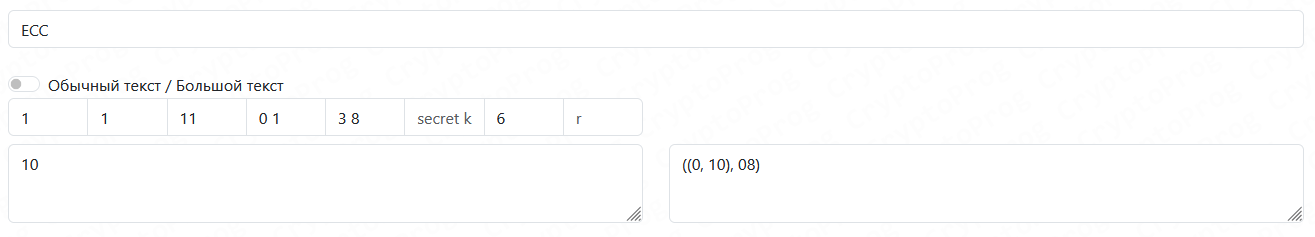
        preparedDecriptedText += alphabetLetter[Number(num)]

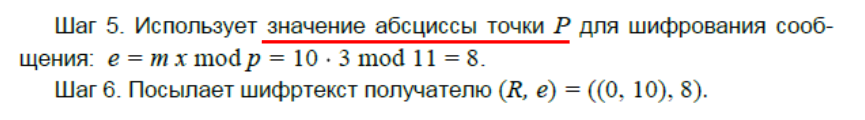
    return preparedDecriptedText

}

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

**ручного теста с указанием ключа)**

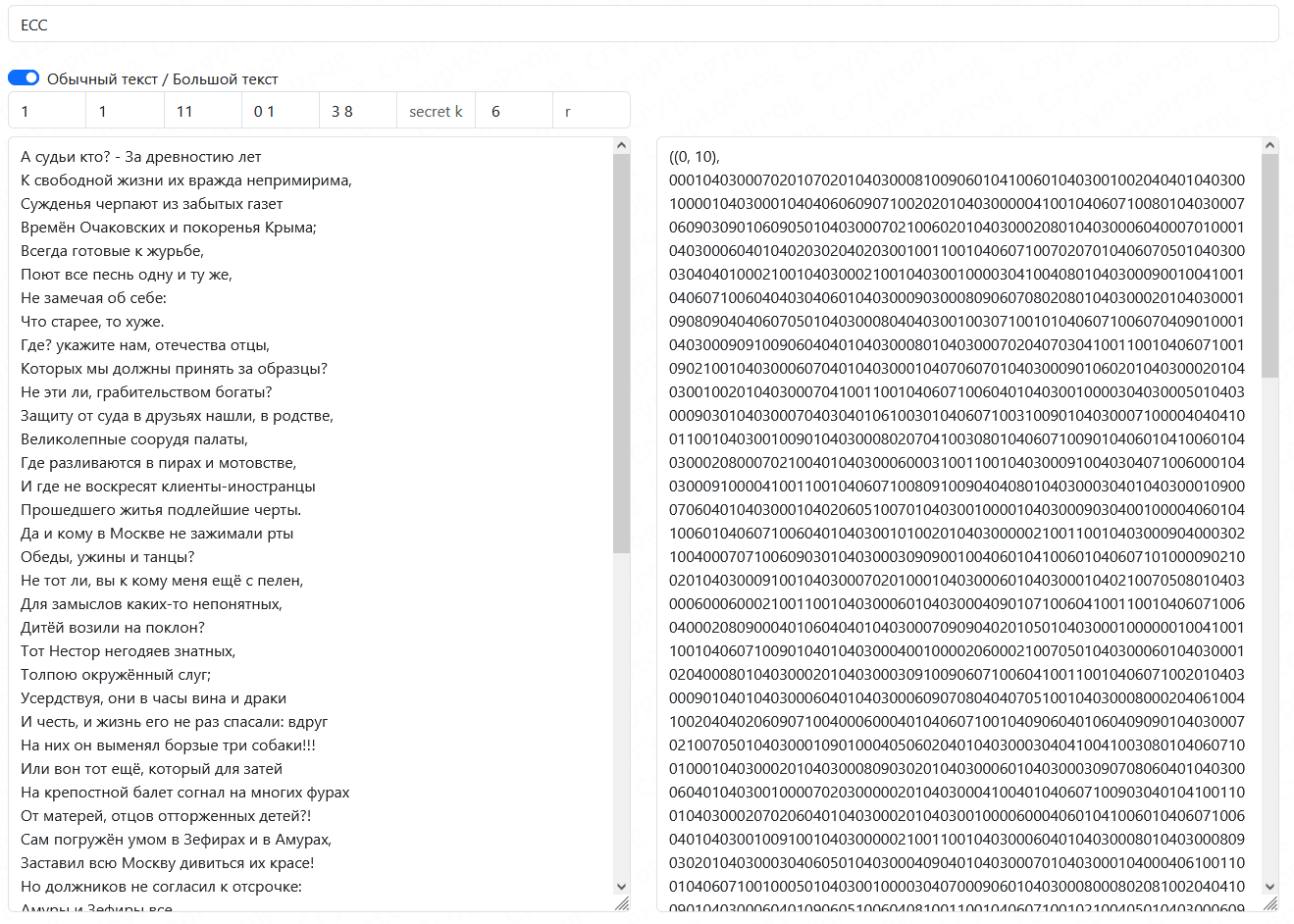
****

****

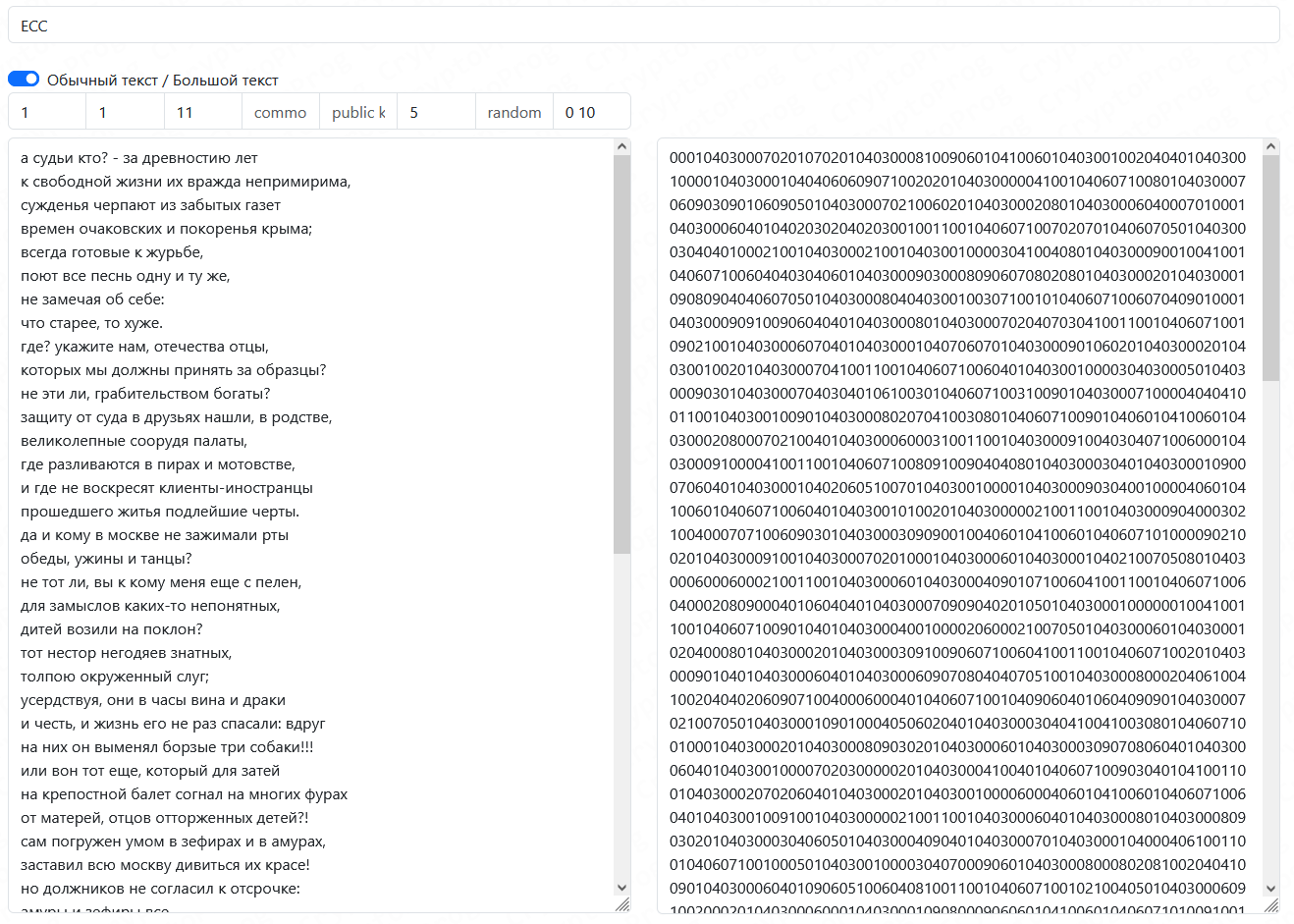
**Работа с текстом не менее 1000 знаков (шифрование и**

**расшифрование с указанием ключа)**

**Зашифрование**

****

**Расшифрование**

****

**Исполняемый файл**

Не существует, есть ссылка <https://cryptoprog.netlify.app/>