**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ**

**Отчёт**

**по математическим и компьютерным основам защиты информации**

Лабораторные работы

Вариант 8

**Выполнил:**

Кендысь Алексей Максимович

студент 3 курса, 7 группы,

специальность

“прикладная математика”

**Преподаватель:**

Ассистент кафедры ММАД ФПМИ,

С.В. Лобач

Минск, 2023

**Содержание:**

Лабораторная работа №1. Классические шифры 2

Условие задачи 2

Теоретические сведения 2

Листинг программы 3

Результаты 5

Лабораторная работа №2. Блочно-итерационные криптосистемы 5

Условие задачи 5

Теоретические сведения 7

Листинг программы 8

Результаты 10

Лабораторная работа №3. Поточные криптосистемы 11

Условие задачи 11

Теоретические сведения 12

Листинг программы 13

Результаты 15

Лабораторная работа №4. Криптосистема RSA 16

Условие задачи 16

Теоретические сведения 16

Листинг программы 17

Результаты 18

Лабораторная работа №5. Схема Эль-Гамаля и хэш-функция SHA-2 19

Условие задачи 19

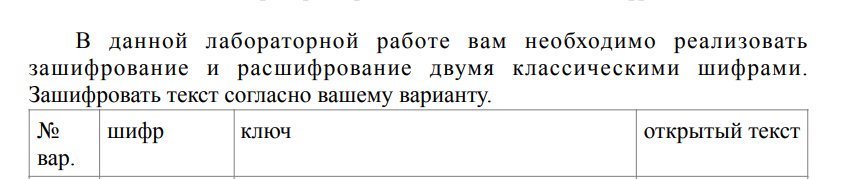
Теоретические сведения 20

Листинг программы 21

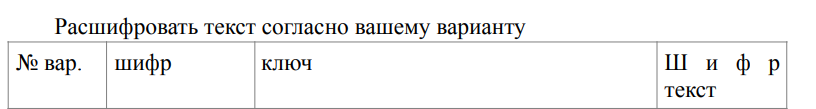
Результаты 24

Лабораторная работа №1. Классические шифры

Условие задачи

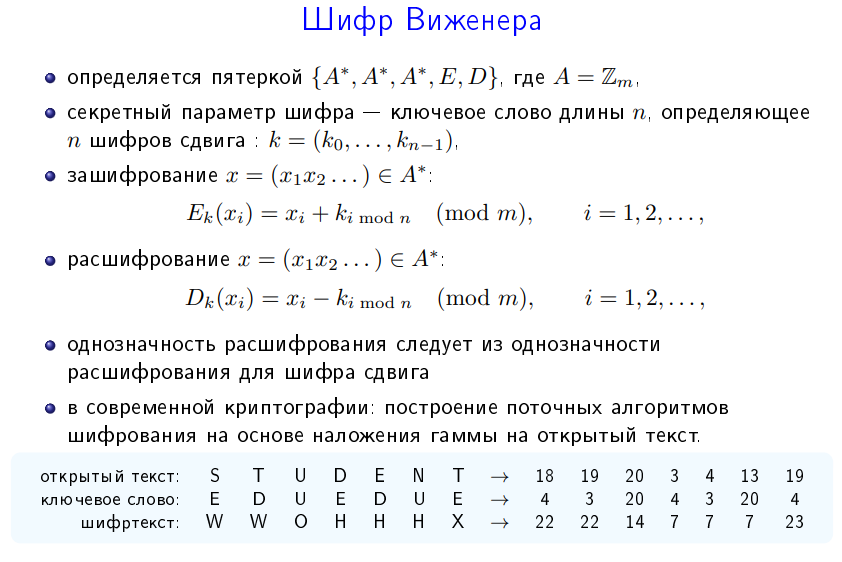


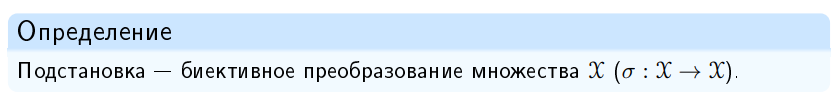


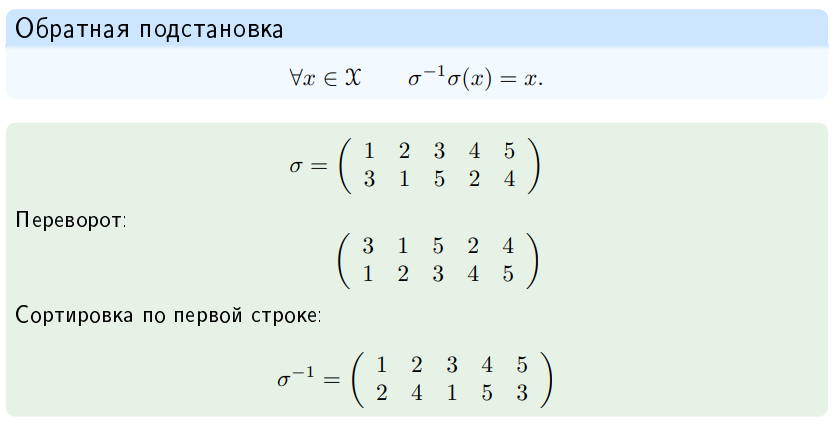


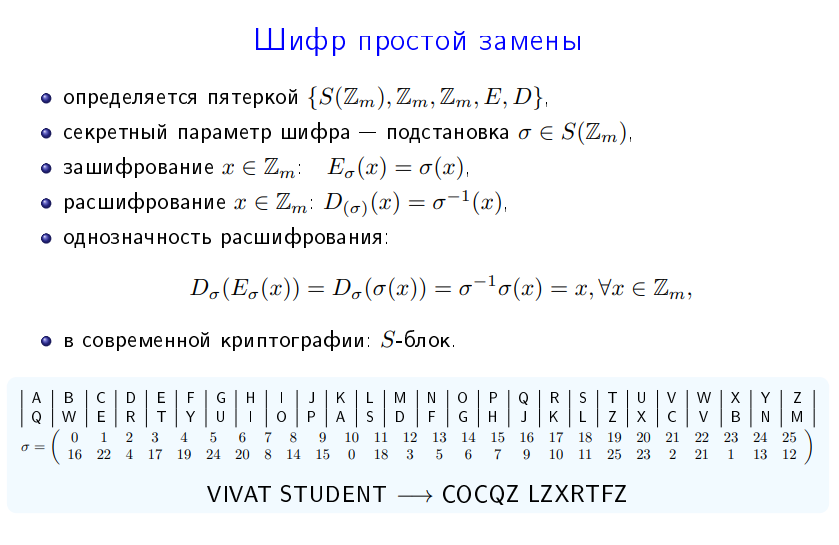


Теоретические сведения





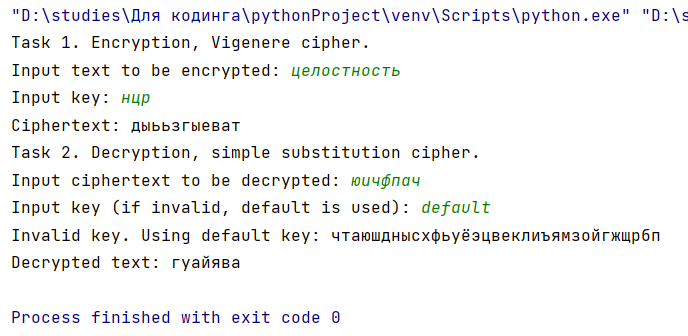




Листинг программы

*# Лабораторная работа №1. Вариант 8. Кендысь Алексей, 3 курс, 7 группа.*alphabet = [  
 'а', 'б', 'в', 'г', 'д', 'е', 'ё', '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'  
]  
  
default\_key2 = 'чтаюшднысхфьуёэцвеклиъямзойгжщрбп'  
  
  
def get\_num(letter):  
 return alphabet.index(letter)  
  
  
def get\_letter(num):  
 return alphabet[num]  
  
  
def input\_text\_key1():  
 in1 = input('Input text to be encrypted: ')  
 in2 = input('Input key: ')  
 return in1.lower(), in2.lower()  
  
  
def input\_text\_key2():  
 in1 = input('Input ciphertext to be decrypted: ')  
 in2 = input('Input key (if invalid, default is used): ')  
  
 if len(in2) != len(alphabet):  
 print('Invalid key. Using default key: ' + default\_key2)  
 in2 = default\_key2  
  
 return in1.lower(), in2.lower()  
  
  
def vigenere():  
 ciphertext\_list = []  
 for i in range(len(plaintext1)):  
 if plaintext1[i] == ' ':  
 ciphertext\_list.append(' ')  
 else:  
 num1 = get\_num(plaintext1[i])  
 num2 = get\_num(key1[i % len(key1)])  
 num = (num1 + num2) % len(alphabet)  
 ciphertext\_list.append(get\_letter(num))  
  
 return ''.join(ciphertext\_list)  
  
  
def get\_sigma():  
 sigma = [[], []]  
 for i in range(len(alphabet)):  
 sigma[0].append(i)  
 key\_num = get\_num(key2[i])  
 sigma[1].append(key\_num)  
  
 return sigma  
  
  
def substitution():  
 sigma = get\_sigma()  
 ciphertext\_list = []  
 for letter in plaintext2:  
 if letter == ' ':  
 ciphertext\_list.append(' ')  
 else:  
 num = get\_num(letter)  
 ind = sigma[1].index(num)  
 cipher\_num = sigma[0][ind]  
 ciphertext\_list += get\_letter(cipher\_num)  
  
 return ''.join(ciphertext\_list)  
  
  
*# main*if \_\_name\_\_ == "\_\_main\_\_":  
 print("Task 1. Encryption, Vigenere cipher.")  
 plaintext1, key1 = input\_text\_key1()  
 print("Ciphertext: " + vigenere())  
 print("Task 2. Decryption, simple substitution cipher.")  
 plaintext2, key2 = input\_text\_key2()  
 print("Decrypted text: " + substitution())

Результаты

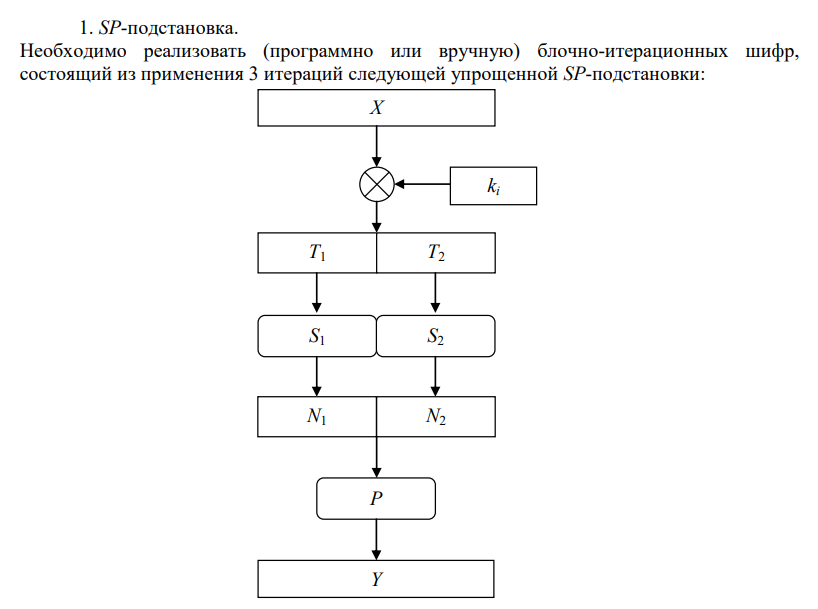


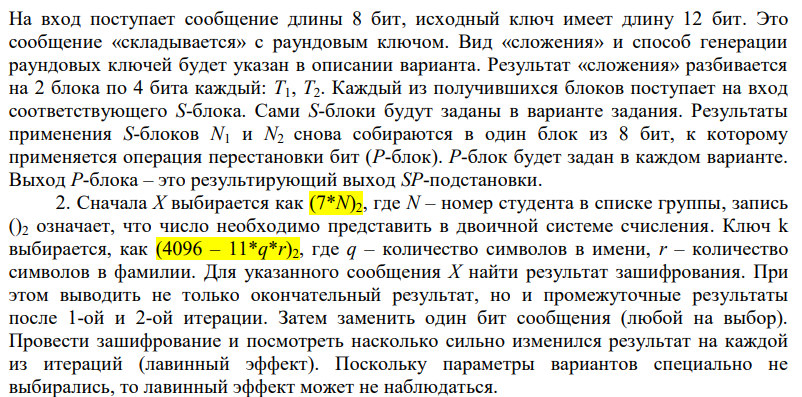
Результат зашифрования слова “целостность” с ключом “нцр” с помощью шифра Виженера: “дыььзгыеват”.

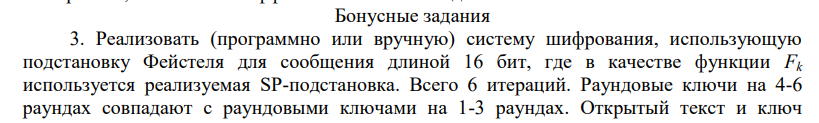
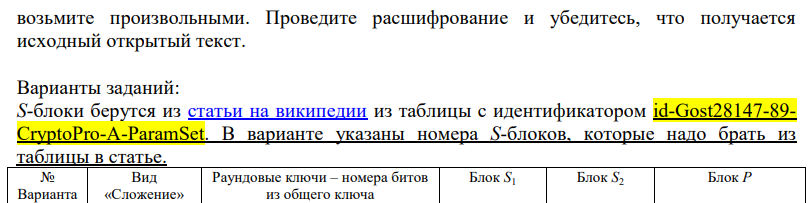
Результат расшифровки текста “юичфпач” с ключом “чтаюшднысхфьуёэцвеклиъямзойгжщрбп” с помощью шифра простой замены: “гуайява”.

Лабораторная работа №2. Блочно-итерационные криптосистемы

Условие задачи

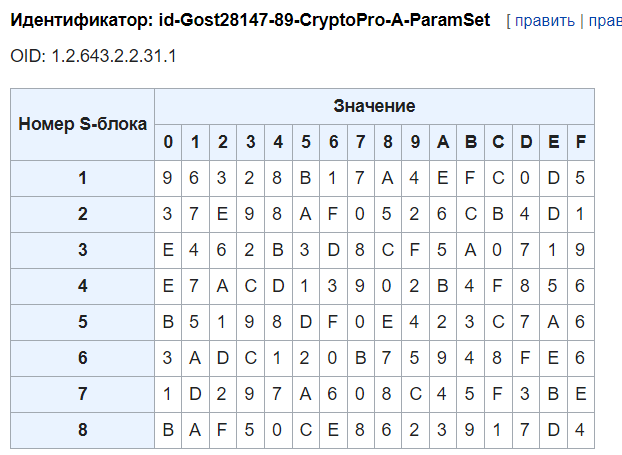




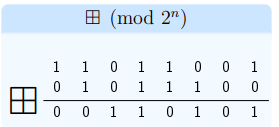
 



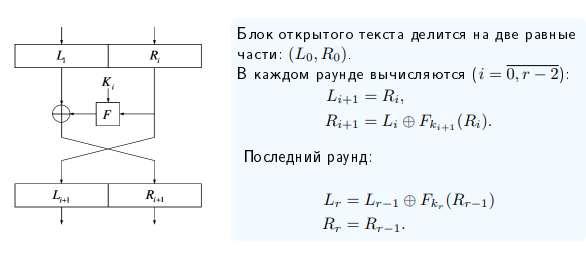
Теоретические сведения

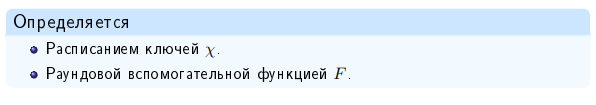


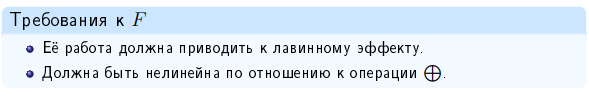
Квадратный плюс – сложение по модулю . Двоичные числа складываются как числа в десятичной системе по модулю , где – количество битов этих чисел. Далее результат сложения переводится обратно в двоичную систему. Пример применения квадратного плюса:



Для бонусного задания, сеть Фейстеля:





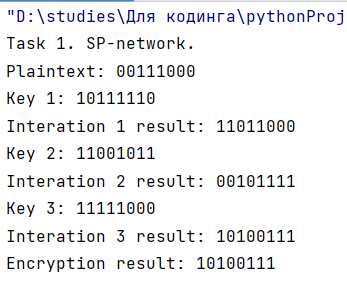


Листинг программы

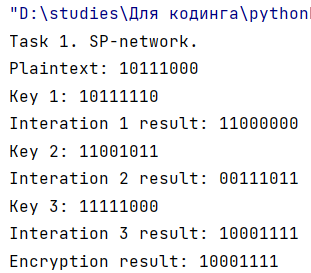
*# Лабораторная работа №2. Вариант 8. Кендысь Алексей, 3 курс, 7 группа.*sp\_x = 0b0011\_1000  
sp\_x\_n = 8  
  
sp\_key = 0b1101\_1110\_0101  
key\_n = 12  
  
key\_set1 = [1, 3, 5, 7, 2, 4, 6, 8]  
key\_set2 = [5, 7, 9, 11, 6, 8, 10, 12]  
key\_set3 = [12, 10, 4, 2, 1, 3, 9, 11]  
key\_set\_list = [key\_set1, key\_set2, key\_set3]  
  
s1 = [11, 5, 1, 9, 8, 13, 15, 0, 14, 4, 2, 3, 12, 7, 10, 6]  
s2 = [14, 7, 10, 12, 13, 1, 3, 9, 0, 2, 11, 4, 15, 8, 5, 6]  
  
p\_shift = 6  
  
sp\_iterations = 3  
  
feistel\_x = 0b0011\_0001\_0100\_0110  
feistel\_x\_n = 16  
  
feistel\_key = 0b1110\_0100\_1000  
  
feistel\_iterations = 6  
  
  
def get\_key(main\_key, key\_set):  
 str\_main\_key = f"{main\_key:0{key\_n}b}"  
 res = []  
 for x in key\_set:  
 res.append(str\_main\_key[x - 1])  
  
 return int(''.join(res), 2)  
  
  
def key\_expansion(main\_key, key\_sets):  
 round\_keys = []  
 for i in range(len(key\_sets)):  
 round\_keys.append(get\_key(main\_key, key\_sets[i]))  
  
 return round\_keys  
  
  
def get\_low\_half(bin\_num, n):  
 length = n >> 1  
 return bin\_num & ((1 << length) - 1)  
  
  
def get\_high\_half(bin\_num, n):  
 length = n >> 1  
 return bin\_num >> length  
  
  
def concat\_bin(bin\_num1, bin\_num2, n):  
 return (bin\_num1 << n) | bin\_num2  
  
  
def circular\_shift\_left(bin\_num, d, n):  
 return ((bin\_num << d) % (1 << n)) | (bin\_num >> (n - d))  
  
  
def sp(x, key):  
 cipher = (x + key) % 256  
 t1 = get\_high\_half(cipher, sp\_x\_n)  
 t2 = get\_low\_half(cipher, sp\_x\_n)  
 n1 = s1[t1]  
 n2 = s2[t2]  
 cipher = concat\_bin(n1, n2, sp\_x\_n >> 1)  
 return circular\_shift\_left(cipher, p\_shift, sp\_x\_n)  
  
  
def sp\_output():  
 print(f"Plaintext: {sp\_x:0{sp\_x\_n}b}")  
  
 sp\_round\_keys = key\_expansion(sp\_key, key\_set\_list)  
 y = 0  
  
 for i in range(sp\_iterations):  
 print(f"Key {i + 1}: {sp\_round\_keys[i]:0{sp\_x\_n}b}")  
 y = sp(sp\_x, sp\_round\_keys[i])  
 print(f"Interation {i + 1} result: {y:0{sp\_x\_n}b}")  
  
 print(f"Encryption result: {y:0{sp\_x\_n}b}")  
  
  
def feistel\_iter(left, right, key):  
 return right, left ^ sp(right, key)  
  
  
def feistel\_encrypt(x, keys):  
 left = get\_high\_half(x, feistel\_x\_n)  
 right = get\_low\_half(x, feistel\_x\_n)  
  
 for i in range(feistel\_iterations):  
 print(f"Left{i}: {left:0{sp\_x\_n}b}. Right{i}: {right:0{sp\_x\_n}b}.")  
 left, right = feistel\_iter(left, right, keys[i])  
  
 left, right = right, left  
 print(f"Left{feistel\_iterations}: {left:0{sp\_x\_n}b}. Right{feistel\_iterations}: {right:0{sp\_x\_n}b}.")  
  
 return concat\_bin(left, right, sp\_x\_n)  
  
  
def feistel\_output():  
 print(f"Plaintext: {feistel\_x:0{feistel\_x\_n}b}")  
  
 feistel\_round\_keys = key\_expansion(feistel\_key, key\_set\_list)  
 feistel\_round\_keys += feistel\_round\_keys  
  
 print("\nEncryption iterations:")  
 cipher = feistel\_encrypt(feistel\_x, feistel\_round\_keys)  
 print(f"\nEncryption result: {cipher:0{feistel\_x\_n}b}")  
  
 feistel\_round\_keys.reverse()  
 print("\nDecryption iterations:")  
 decrypted\_text = feistel\_encrypt(cipher, feistel\_round\_keys)  
 print(f"\nDecryption result: {decrypted\_text:0{feistel\_x\_n}b}")  
  
  
*# main*if \_\_name\_\_ == "\_\_main\_\_":  
 print("Task 1. SP-network.")  
 sp\_output()  
  
 print("\nTask 2. Feistel-network.")  
 feistel\_output()

Результаты

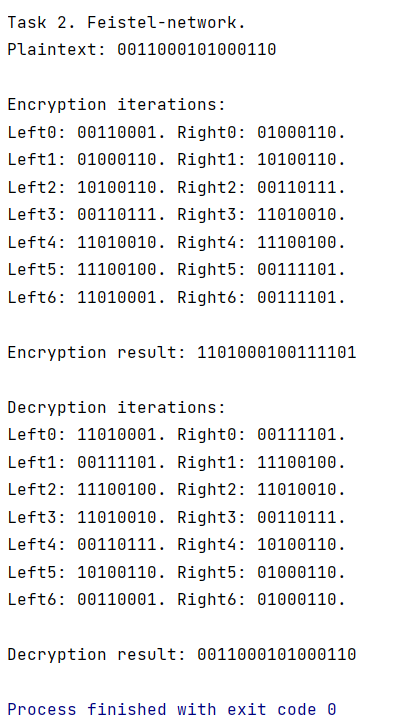
SP-подстановка для заданного открытого текста:



SP-подстановка при изменении первого бита исходного открытого текста:



Бонусное задание:

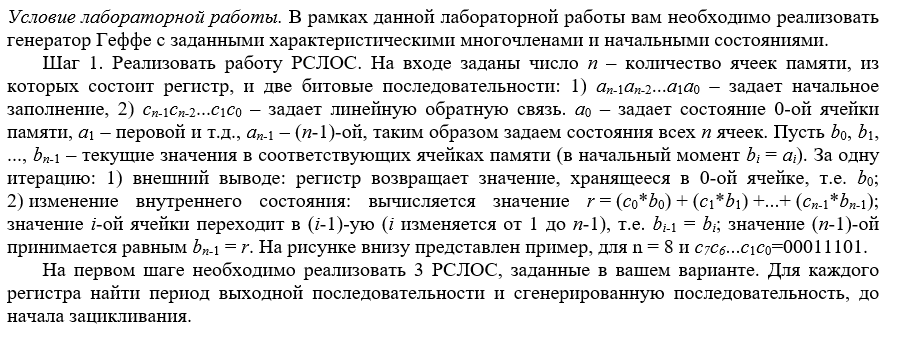


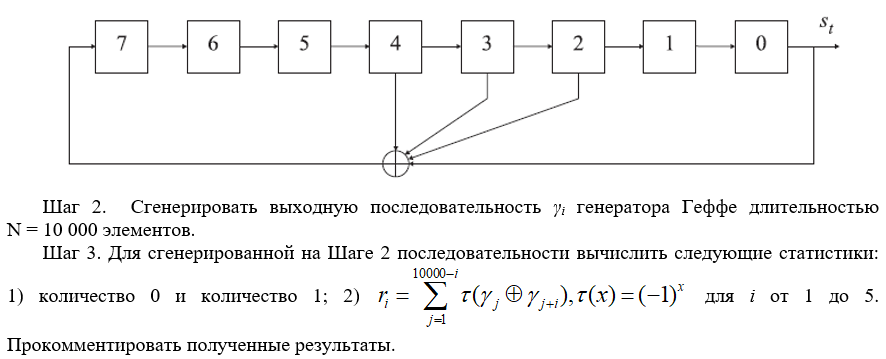
При использовании SP-подстановки для открытого текста “00111000” на 1-ой итерации получили “11011000”, на 2-ой – “00101111”, на 3-ей – “10100111”. Т.е. в результате получили шифртекст – “10100111”. При изменении одного бита в открытом тексте шифр изменился только на два бита на каждой итерации. Таким образом, наблюдается довольно малый лавинный эффект.

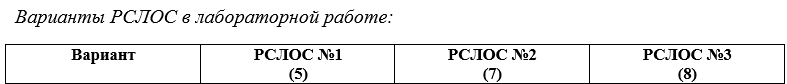
В бонусном задании для открытого текста “0011000101000110” использовался алгоритм шифрования, основанный на сети Фейстеля. В результате был получен шифртекст “1101000100111101”. Для проверки результата была произведена расшифровка, в результате которой получили исходный открытый текст, т.е. сеть работает правильно.

Лабораторная работа №3. Поточные криптосистемы

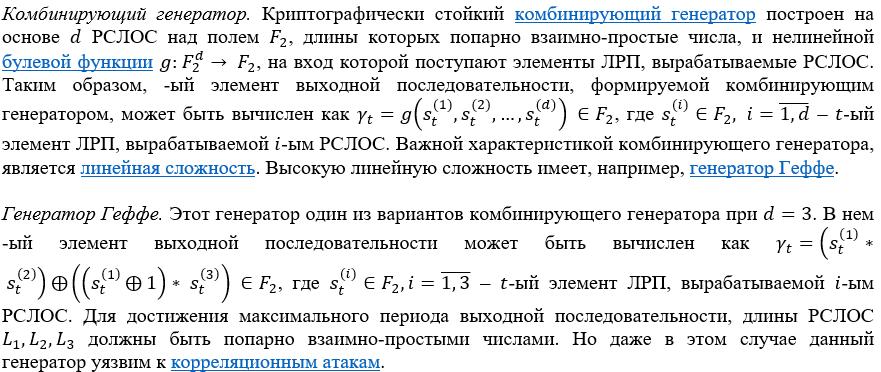
Условие задачи

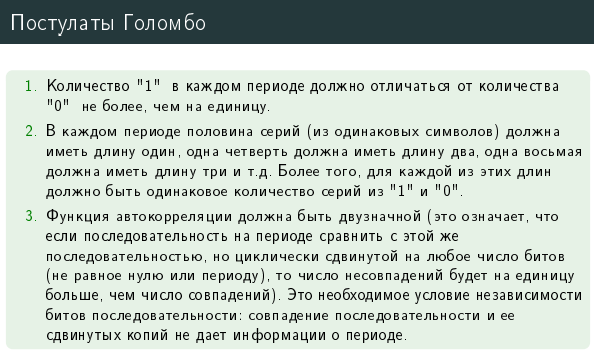




Теоретические сведения

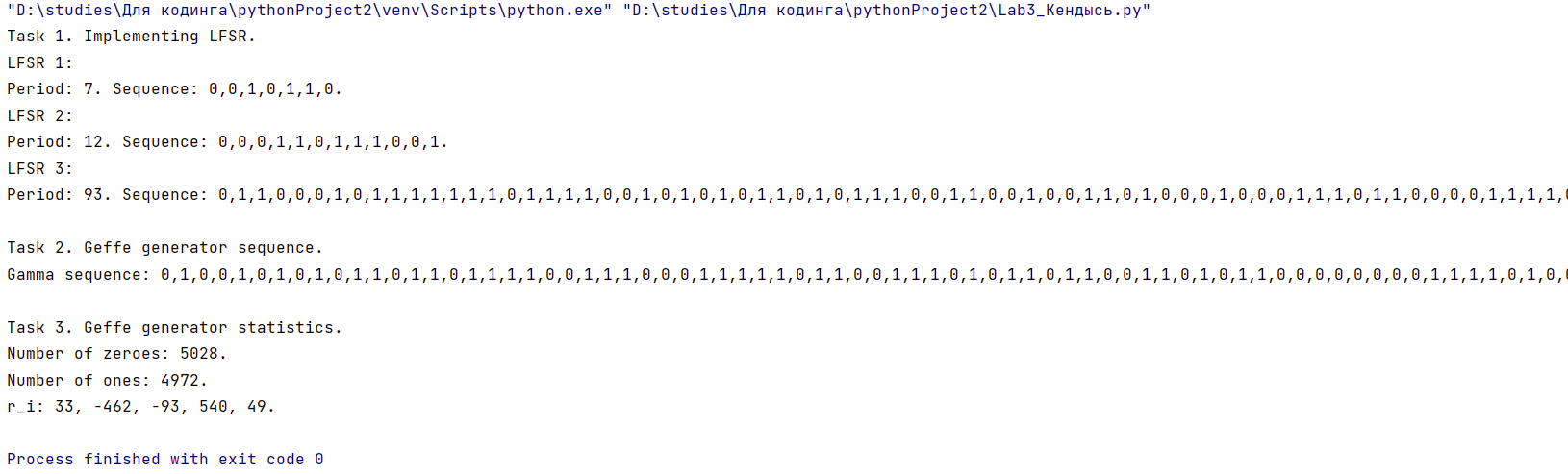




Листинг программы

*# Лабораторная работа №3. Вариант 8. Кендысь Алексей, 3 курс, 7 группа.*n1 = 5  
a1 = 0b10100  
c1 = 0b11001  
  
n2 = 7  
a2 = 0b1011000  
c2 = 0b1000011  
  
n3 = 8  
a3 = 0b01000110  
c3 = 0b00001101  
  
N = 10000  
i\_n = 5  
  
  
def xor\_number(bin\_num):  
 return bin\_num.bit\_count() % 2  
  
  
def gamma\_xor(gamma1, gamma2):  
 if gamma1 ^ gamma2 == 0:  
 return 1  
 else:  
 return -1  
  
  
class LFSR:  
 n = 0  
 a, b, c = 0b0, 0b0, 0b0  
  
 def \_\_init\_\_(self, n, a, c):  
 self.n = n  
 self.a = a  
 self.c = c  
  
 self.b = a  
 self.period, self.seq = self.get\_period\_and\_sequence()  
  
 def get\_period\_and\_sequence(self):  
 res = 0  
 seq = []  
 while True:  
 seq.append(self.b & 0b1)  
  
 s = self.b & self.c  
 r = xor\_number(s)  
 self.b = (r << (self.n - 1)) | (self.b >> 1)  
  
 res += 1  
 if self.b == self.a:  
 break  
  
 return res, seq  
  
  
class GeffeGenerator:  
 n = 0  
 gamma\_seq = []  
 lfsr\_seq = []  
  
 def \_\_init\_\_(self, n, lfsr\_seq):  
 self.n = n  
 self.lfsr\_seq = lfsr\_seq  
  
 self.gamma\_seq = self.get\_gamma\_seq()  
  
 def get\_gamma\_seq(self):  
 res = []  
  
 seq1 = self.lfsr\_seq[0].seq  
 seq2 = self.lfsr\_seq[1].seq  
 seq3 = self.lfsr\_seq[2].seq  
  
 per1 = self.lfsr\_seq[0].period  
 per2 = self.lfsr\_seq[1].period  
 per3 = self.lfsr\_seq[2].period  
  
 for t in range(self.n):  
 gamma = (seq1[t % per1] & seq2[t % per2]) ^ ((seq1[t % per1] ^ 0b1) & seq3[t % per3])  
 res.append(gamma)  
  
 return res  
  
 def count\_ones(self):  
 return sum(self.gamma\_seq)  
  
 def get\_r\_sum(self, i):  
 r\_sum = 0  
 for t in range(self.n - i):  
 r\_sum += gamma\_xor(self.gamma\_seq[t], self.gamma\_seq[t + i])  
 return r\_sum  
  
 def r\_stat(self, i\_n1):  
 res = []  
 for i in range(1, i\_n1 + 1):  
 res.append(self.get\_r\_sum(i))  
  
 return res  
  
  
def lfsr\_output(lfsr\_seq):  
 for i in range(len(lfsr\_seq)):  
 print(f"LFSR {i + 1}:")  
 seq\_string = ','.join(map(str, lfsr\_seq[i].seq))  
 print(f"Period: {lfsr\_seq[i].period}. Sequence: {seq\_string}.")  
  
  
def geffe\_seq\_output(geffe\_seq):  
 print(f"Gamma sequence: {','.join(map(str, geffe\_seq))}.")  
  
  
def geffe\_stats\_output(zeroes, ones, r\_stat):  
 print(f"Number of zeroes: {zeroes}.")  
 print(f"Number of ones: {ones}.")  
 print(f"r\_i: {', '.join(map(str, r\_stat))}.")  
  
  
*# main*if \_\_name\_\_ == "\_\_main\_\_":  
 print("Task 1. Implementing LFSR.")  
 lfsr1 = LFSR(n1, a1, c1)  
 lfsr2 = LFSR(n2, a2, c2)  
 lfsr3 = LFSR(n3, a3, c3)  
 lfsr\_list = [lfsr1, lfsr2, lfsr3]  
  
 lfsr\_output(lfsr\_list)  
  
 print("\nTask 2. Geffe generator sequence.")  
 geffe = GeffeGenerator(N, lfsr\_list)  
 geffe\_seq\_output(geffe.gamma\_seq)  
  
 print("\nTask 3. Geffe generator statistics.")  
 one\_sum = geffe.count\_ones()  
 geffe\_stats\_output(N - one\_sum, one\_sum, geffe.r\_stat(i\_n))

Результаты



РСЛОС 1:

Период 7. Последовательность: 0,0,1,0,1,1,0.

РСЛОС 2:

Период 12. Последовательность: 0,0,0,1,1,0,1,1,1,0,0,1.

РСЛОС 3:

Период 93. Последовательность: 0,1,1,0,0,0,1,0,1,1,1,1,1,1,1,0,1,1,1,1,0,0,1,0,1,0,1,0,1,1,0,1,0,1,1,1,0,0,1,1,0,0,1,0,0,1,1,0,1,0,0,0,1,0,0,0,1,1,1,0,1,1,0,0,0,0,1,1,1,1,0,1,0,0,1,0,0,0,0,0,1,0,1,0,0,1,1,1,0,0,0,0,0.

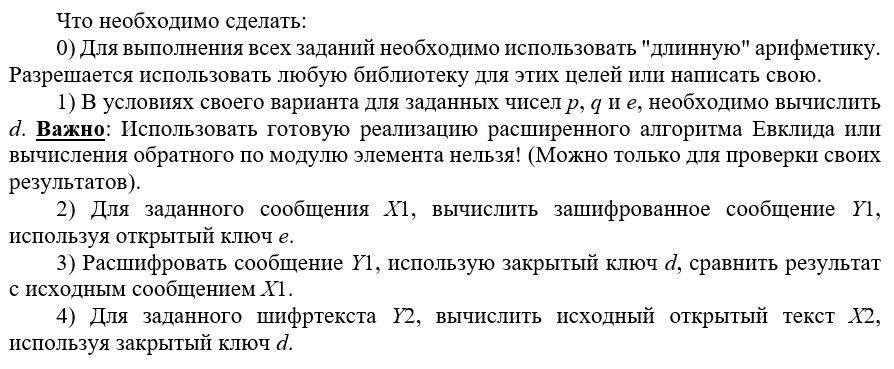
Первые значения генератора Гёффе (всего их 10000): 0,1,0,0,1,0,1,0,1,0,1,1,0,1,1,0,1,1,1,1,0,0,1,1,1,0,0,0,1,1,1,1,1,0,1,1,0,0,1,1,1,0,1,0,1,1,0,1,1,0,0,1,1,0,1,0,1,1,0,0,0,0,0,0,0,0,1,1,1,1,0,1,0,0,0,1,0, ... .

Получили 5028 нулей и 4972 единиц в последовательности, что довольно хорошо. Но всё же количество “0” отличается от количества ”1” больше, чем на единицу, поэтому первый постулат Голомба не выполняется, хоть значения и близки.

Статистика описывает третий постулат Голомба. При значение статистики двухзначное, т.е. постулат выполняется, это преимущество последовательности. Для же он не выполнен, т.е. есть и недостатки (небольшая зависимость между значениями).

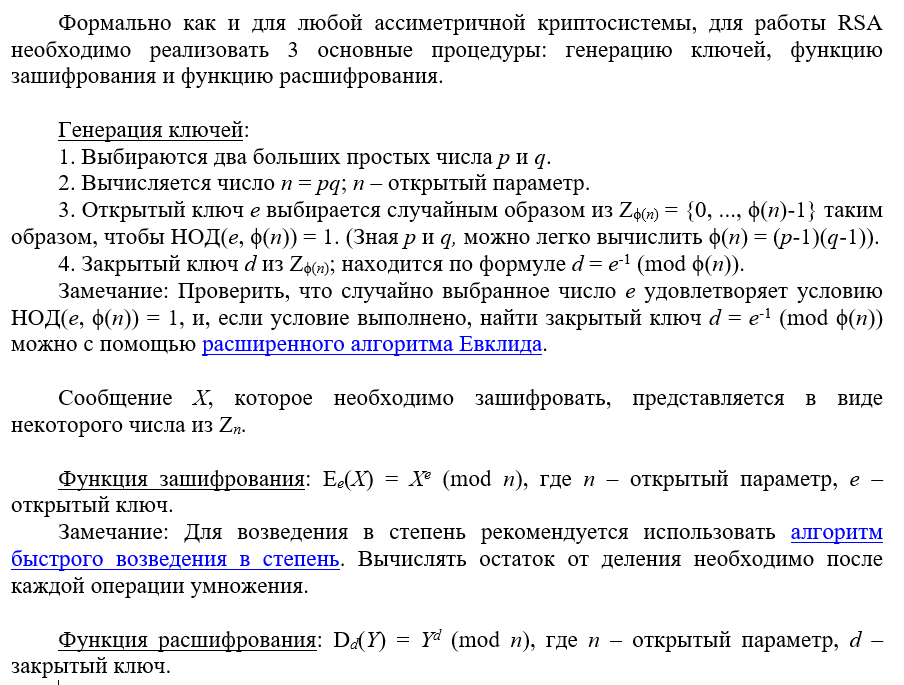
Лабораторная работа №4. Криптосистема RSA

Условие задачи



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № Вар. | *p* | *q* | *e* | *X*1 | *Y*2 |
| 8 | 1102914252601991 | 571301412050021 | 624840313709071966800768010501 | 267222621555915275276288463243 | 291064433434228628162063527294 |

Теоретические сведения

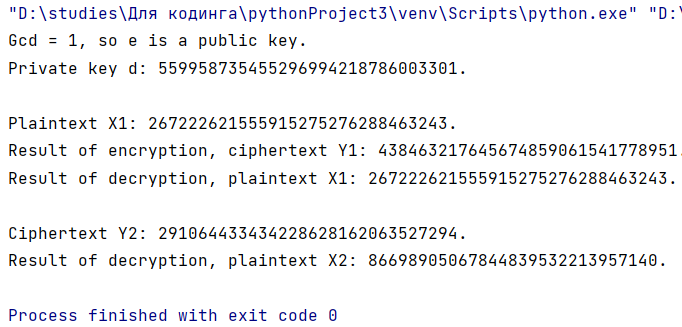




Листинг программы

*# Лабораторная работа №4. Вариант 8. Кендысь Алексей, 3 курс, 7 группа.*p = 1102914252601991  
q = 571301412050021  
n = p \* q  
e = 624840313709071966800768010501  
  
x1 = 267222621555915275276288463243  
  
y2 = 291064433434228628162063527294  
  
  
def extended\_euclidian(a, b):  
 if a == 0:  
 return 0, 1, b  
  
 prev\_x, x = 1, 0  
 prev\_y, y = 0, 1  
  
 while b != 0:  
 div = a // b  
  
 (x, prev\_x) = (prev\_x - (div \* x), x)  
 (y, prev\_y) = (prev\_y - (div \* y), y)  
  
 (a, b) = (b, a % b)  
  
 return prev\_x, prev\_y, a  
  
  
def fast\_pow(num, power, mod):  
 num %= mod  
 res = 1  
  
 while power > 0:  
 if power % 2 == 1:  
 res \*= num  
 res %= mod  
  
 num \*= num  
 num %= mod  
 power //= 2  
  
 return res  
  
  
def get\_d():  
 phi = (p - 1) \* (q - 1)  
 res, unused, gcd = extended\_euclidian(e, phi)  
  
 if res < 0:  
 res += phi  
  
 if gcd == 1:  
 print("Gcd = 1, so e is a public key.")  
 return res  
 else:  
 print("e is not a public key.")  
 return "Error"  
  
  
def encrypt(x):  
 return fast\_pow(x, e, n)  
  
  
def decrypt(y):  
 return fast\_pow(y, d, n)  
  
  
*# main*if \_\_name\_\_ == "\_\_main\_\_":  
 d = get\_d()  
 print(f"Private key d: {int(d)}.")  
  
 y1 = encrypt(x1)  
 verif\_x1 = decrypt(y1)  
 print(f"\nPlaintext X1: {int(x1)}.")  
 print(f"Result of encryption, ciphertext Y1: {int(y1)}.")  
 print(f"Result of decryption, plaintext X1: {int(verif\_x1)}.")  
  
 x2 = decrypt(y2)  
 print(f"\nCiphertext Y2: {int(y2)}.")  
 print(f"Result of decryption, plaintext X2: {int(x2)}.")

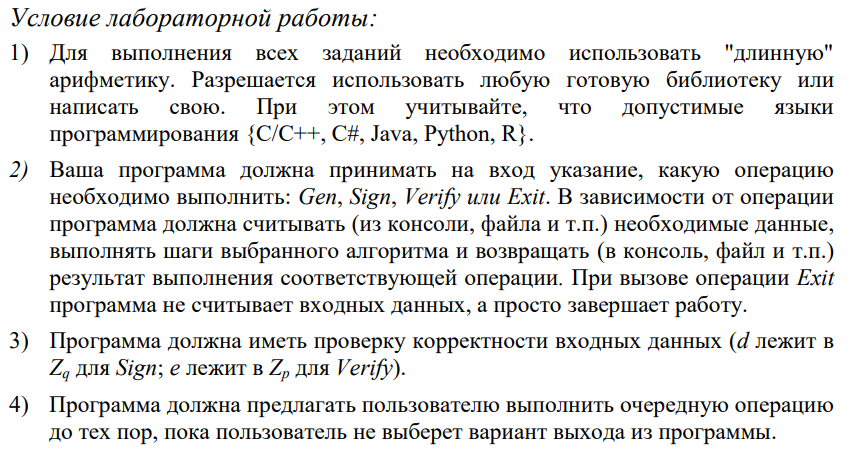
Результаты

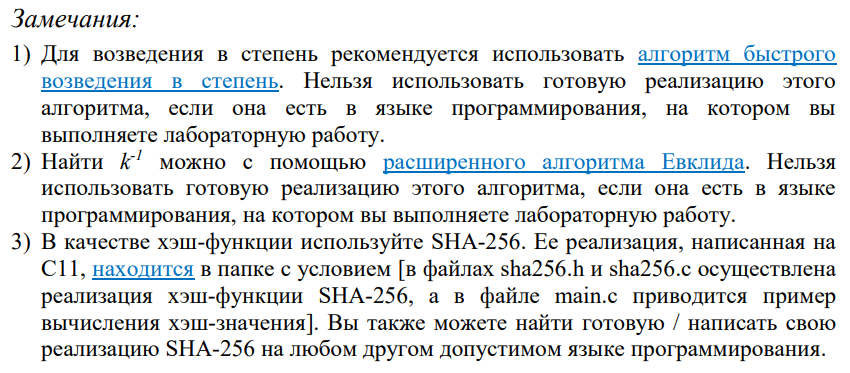


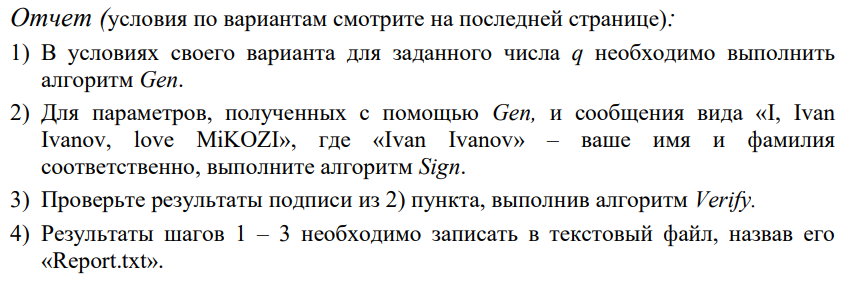
Проверка, что является открытым ключом, прошла успешно (соответствующий НОД = 1). Для открытого текста = 267222621555915275276288463243получили шифртекст = 438463217645674859061541778951 Также было произведена расшифровка для проверки ответа, результат совпал с исходным открытым текстом, алгоритм работает корректно. Для шифртекста = 291064433434228628162063527294 получили с помощью расшифровки открытый текст = 86698905067844839532213957140.

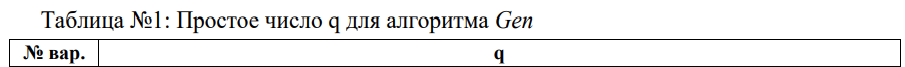
Лабораторная работа №5. Схема Эль-Гамаля и хэш-функция SHA-2

Условие задачи



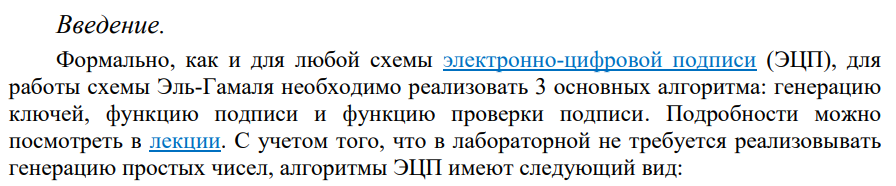


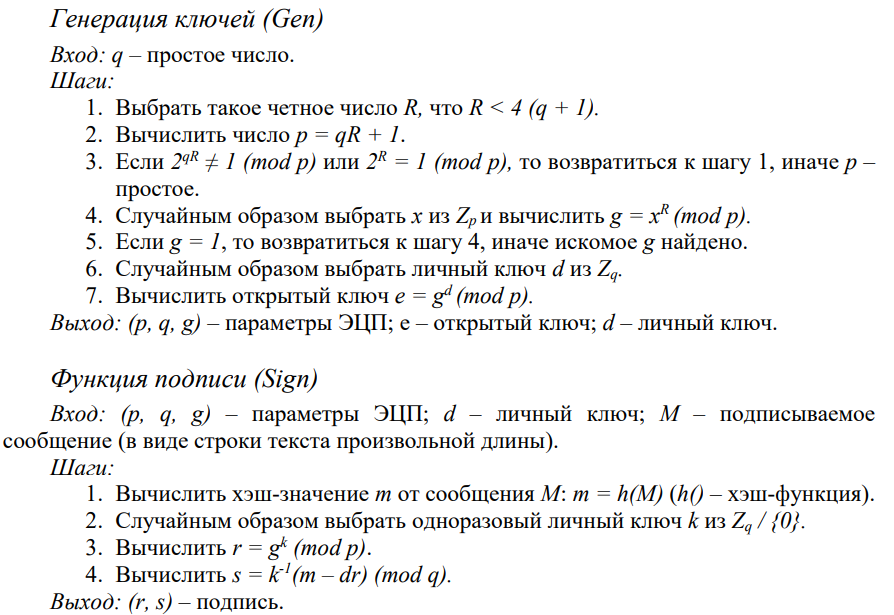


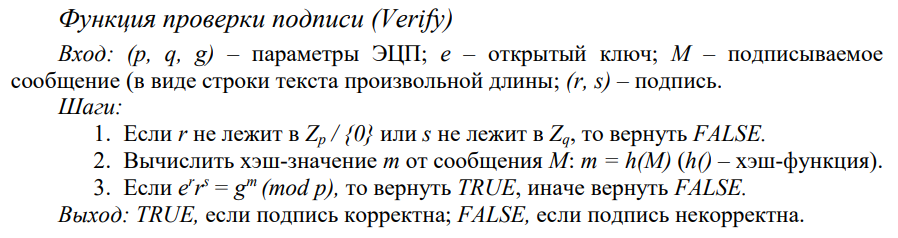


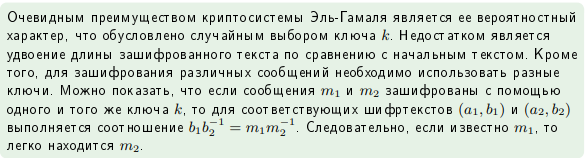


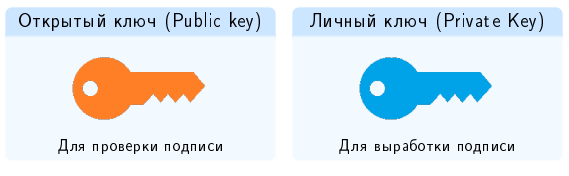
Теоретические сведения









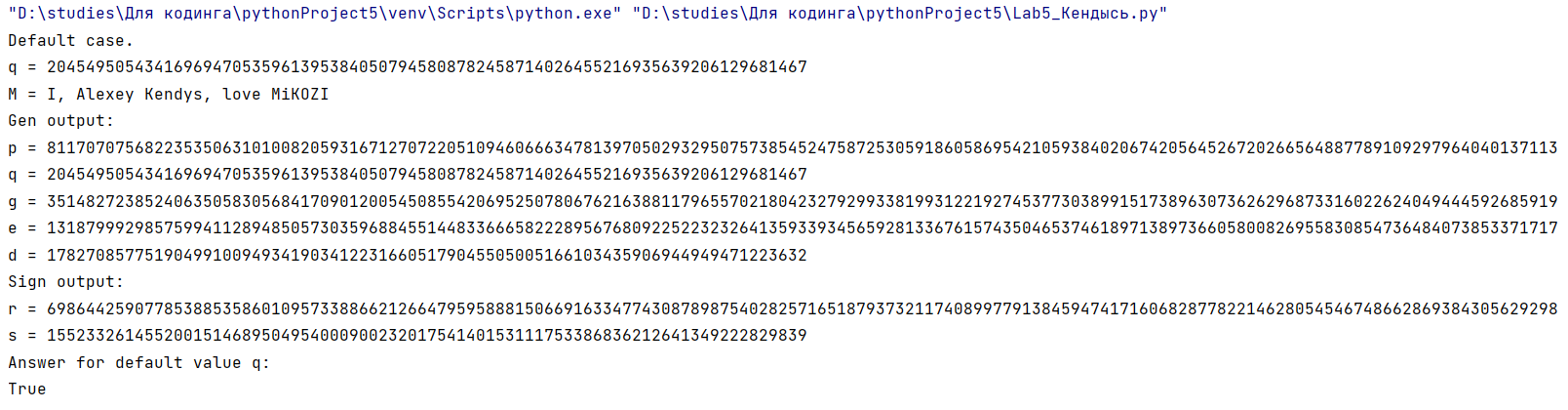


Листинг программы

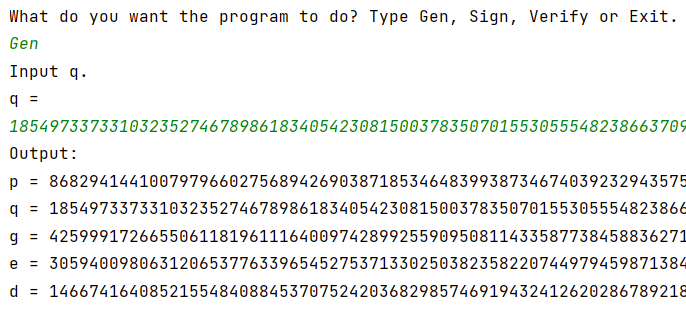
*# Лабораторная работа №5. Вариант 8. Кендысь Алексей, 3 курс, 7 группа.*import random  
from hashlib import sha256  
  
q = 204549505434169694705359613953840507945808782458714026455216935639206129681467  
m = "I, Alexey Kendys, love MiKOZI"  
  
  
def extended\_euclidian(a, b):  
 if a == 0:  
 return 0, 1, b  
  
 prev\_x, x = 1, 0  
 prev\_y, y = 0, 1  
  
 while b != 0:  
 div = a // b  
  
 (x, prev\_x) = (prev\_x - (div \* x), x)  
 (y, prev\_y) = (prev\_y - (div \* y), y)  
  
 (a, b) = (b, a % b)  
  
 return prev\_x, prev\_y, a  
  
  
def get\_reverse(num, mod):  
 res, unused, gcd = extended\_euclidian(num, mod)  
  
 if res < 0:  
 res += mod  
  
 return res  
  
  
def fast\_pow(num, power, mod):  
 num %= mod  
 res = 1  
  
 while power > 0:  
 if power % 2 == 1:  
 res \*= num  
 res %= mod  
  
 num \*= num  
 num %= mod  
 power //= 2  
  
 return res  
  
  
def gen(in\_q):  
 while True:  
 in\_r = random.randrange(2, 4 \* (in\_q + 1), 2)  
 out\_p = in\_q \* in\_r + 1  
 if fast\_pow(2, in\_q \* in\_r, out\_p) == 1 and fast\_pow(2, in\_r, out\_p) != 1:  
 break  
  
 while True:  
 x = random.randrange(out\_p)  
 out\_g = fast\_pow(x, in\_r, out\_p)  
 if out\_g != 1:  
 break  
  
 out\_d = random.randrange(in\_q)  
 out\_e = fast\_pow(out\_g, out\_d, out\_p)  
  
 return (out\_p, in\_q, out\_g), out\_e, out\_d  
  
  
def sign(in\_p, in\_q, in\_g, in\_d, in\_m):  
 h = sha256(in\_m.encode("utf8"))  
 m1 = int(h.hexdigest(), base=16)  
  
 k = random.randrange(1, in\_q)  
 out\_r = fast\_pow(in\_g, k, in\_p)  
 out\_s = (get\_reverse(k, in\_q) \* ((m1 - in\_d \* out\_r) % in\_q)) % in\_q  
  
 return out\_r, out\_s  
  
  
def verify(in\_p, in\_q, in\_g, in\_e, in\_m, in\_r, in\_s):  
 if in\_r >= in\_p or in\_r <= 0 or in\_s >= in\_q or in\_s < 0:  
 return False  
  
 h = sha256(in\_m.encode("utf8"))  
 m1 = int(h.hexdigest(), base=16)  
  
 if (fast\_pow(in\_e, in\_r, in\_p) \* fast\_pow(in\_r, in\_s, in\_p)) % in\_p == fast\_pow(in\_g, m1, in\_p):  
 return True  
 else:  
 return False  
  
  
def input\_int\_value(value\_name):  
 print(f"{value\_name} = ")  
 value = input()  
  
 return int(value)  
  
  
def input\_str\_value(value\_name):  
 print(f"{value\_name} = ")  
 value = input()  
  
 return value  
  
  
def menu():  
 while True:  
 print("\nWhat do you want the program to do? Type Gen, Sign, Verify or Exit.")  
 inquiry = input()  
 if inquiry == 'Gen':  
 print("Input q.")  
 in\_q = input\_int\_value("q")  
  
 (in\_p, in\_q, in\_g), in\_e, in\_d = gen(in\_q)  
 print("Output:")  
 print(f"p = {in\_p}\nq = {in\_q}\ng = {in\_g}\ne = {in\_e}\nd = {in\_d}")  
  
 elif inquiry == 'Sign':  
 print("Input p, q, g, d, M.")  
  
 in\_p = input\_int\_value("p")  
 in\_q = input\_int\_value("q")  
 in\_g = input\_int\_value("g")  
 in\_d = input\_int\_value("d")  
 in\_m = input\_str\_value("M")  
  
 in\_r, in\_s = sign(in\_p, in\_q, in\_g, in\_d, in\_m)  
 print("Output:")  
 print(f"r = {in\_r}\ns = {in\_s}")  
  
 elif inquiry == 'Verify':  
 print("Input p, q, g, e, M, r, s.")  
  
 in\_p = input\_int\_value("p")  
 in\_q = input\_int\_value("q")  
 in\_g = input\_int\_value("g")  
 in\_e = input\_int\_value("e")  
 in\_m = input\_str\_value("M")  
 in\_r = input\_int\_value("r")  
 in\_s = input\_int\_value("s")  
  
 print("Output:")  
 print(verify(in\_p, in\_q, in\_g, in\_e, in\_m, in\_r, in\_s))  
  
 elif inquiry == 'Exit':  
 break  
 else:  
 print("Invalid input.")  
  
  
*# main*if \_\_name\_\_ == "\_\_main\_\_":  
 print("Default case.")  
 print(f"q = {q}")  
 print(f"M = {m}")  
  
 (p, q, g), e, d = gen(q)  
 print("Gen output:")  
 print(f"p = {p}")  
 print(f"q = {q}")  
 print(f"g = {g}")  
 print(f"e = {e}")  
 print(f"d = {d}")  
  
 r, s = sign(p, q, g, d, m)  
 print("Sign output:")  
 print(f"r = {r}")  
 print(f"s = {s}")  
  
 print("Answer for default value q:")  
 print(verify(p, q, g, e, m, r, s))  
  
 menu()

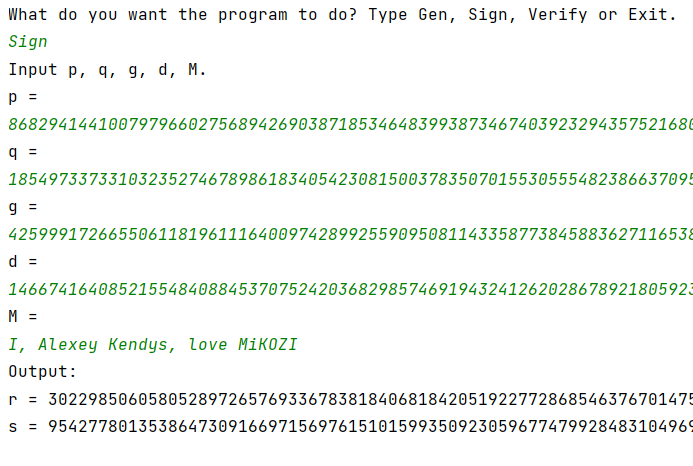
Результаты

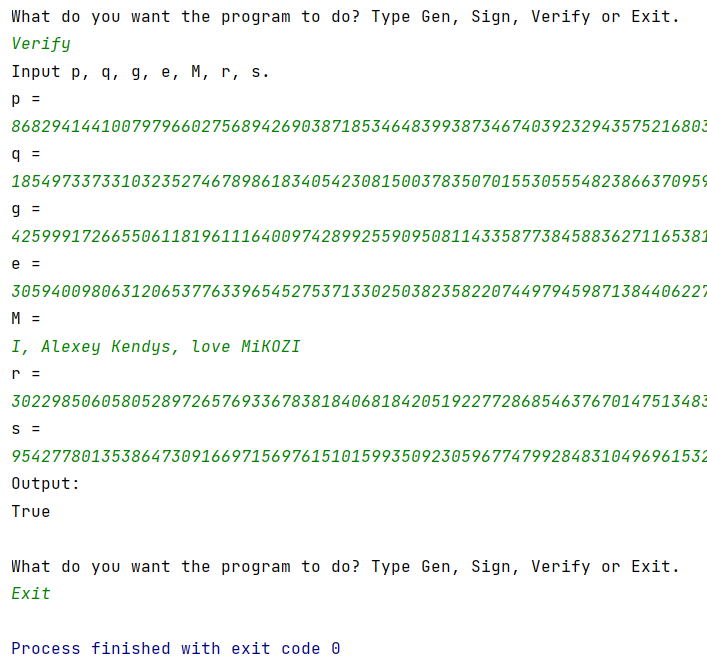
Проверка выполнения алгоритма для данных индивидуального варианта:



Проверка работы меню, или, другими словами, возможности для пользователя выбрать нужную функцию и ввести свои данные (используются данные из 7-го варианта):







Функция Verify в обоих случаях выдала True, т.е. алгоритм работает корректно. Был успешно сгенерирован открытый ключ (public key) и личный ключ (private key) . Для сообщения с помощью личного ключа была выработана подпись , а далее проверена на корректность.