**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ**

**Кафедра теории вероятностей и математической статистики**

**ОТЧЕТ**

по лабораторным работам

учебной дисциплины

«Математические и компьютерные основы защиты информации»

Вариант № 1

**Выполнил:**

Белоушко Степан Игоревич,

3 курс 7а группа, специальность «прикладная математика»

**Преподаватель:**

Лобач Сергей Викторович

Минск, 2023

**Лабораторная работа 1**

Теория

Ключом шифра служит перемешанный произвольным образом алфавит. Например, ключом может быть следующая последовательность букв: XFQABOLYWJGPMRVIHUSDZKNTEC.

При шифровании каждая буква в тексте заменяется по следующему правилу. Первая буква алфавита замещается первой буквой ключа, вторая буква алфавита — второй буквой ключа и так далее. В нашем примере буква A будет заменена на X, буква B на F.

При расшифровке буква сперва ищется в ключе и затем заменяется буквой, стоящей в алфавите на той же позиции.

Аффиный шифр тоже реализует простую подстановку, но обеспечивает немного большее пространство ключей по сравнению с шифром Цезаря. В аффинном шифре каждой букве алфавита размера m ставится в соответствие число из диапазона 0… m-1. Затем при помощи специальной формулы, вычисляется новое число, которое заменит старое в шифртексте.

Процесс шифрования можно описать следующей формулой:

image

где x — номер шифруемой буквы в алфавите; m — размер алфавита; a, b — ключ шифрования.

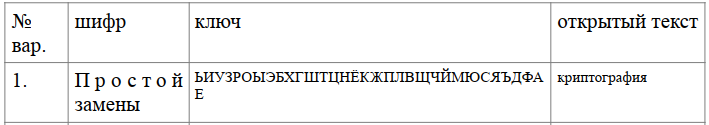
Для расшифровки вычисляется другая функция:

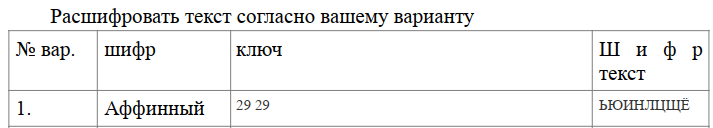
image

где a-1 — число обратное a по модулю m. Это значит, что для корректной расшифровки число a должно быть взаимно простым с m.

Задание

Реализовать зашифрование и расшифрование двумя классическими шифрами. Зашифровать текст согласно вашему варианту.





Код

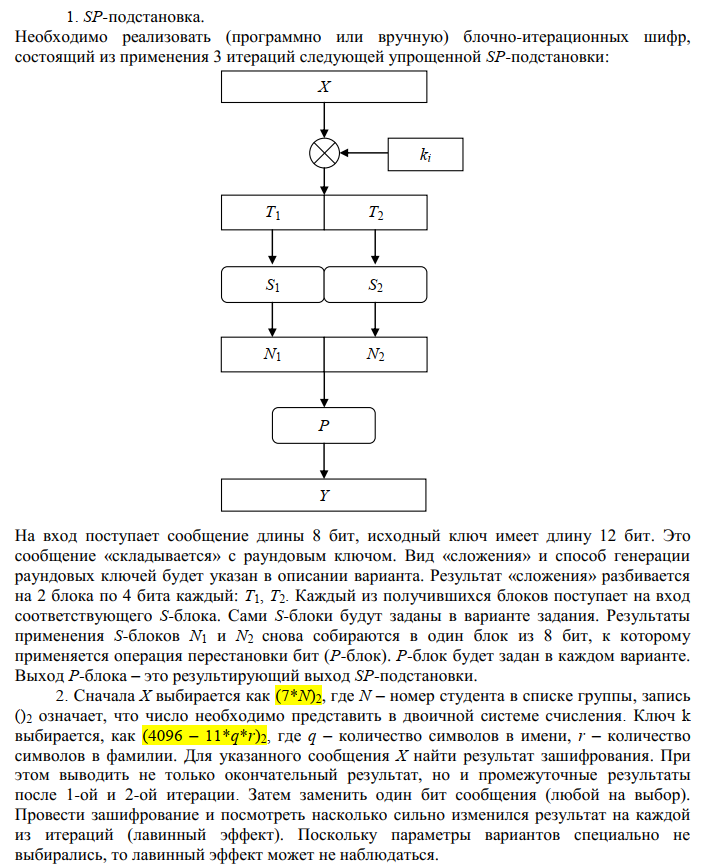
import java.util.Arrays;  
import java.util.List;  
  
public class Main {  
 public static void main(String[] args) {  
 String[] alphabet = new String[] {"а", "б", "в", "г", "д", "е", "ё", "ж", "з", "и", "й", "к",  
 "л", "м", "н", "о", "п", "р", "с", "т", "у", "ф", "х", "ц", "ч", "ш", "щ", "ъ", "ы", "ь", "э", "ю", "я"};  
 // абвгдеёжзийклмнопрстуфхцчшщъыьэюя  
 String [] cipher = "ьиузроыэбхгштцнёкжплвщчймюсяъдфае".split("");  
 String[] text = "криптография".split("");  
 List<String> alphabet2 = Arrays.*asList*(alphabet);  
 StringBuilder sb = new StringBuilder();  
 for(int i = 0; i < text.length; i++) {  
 sb.append(cipher[alphabet2.indexOf(text[i])]);  
 }  
 String cipherText = sb.toString();  
 System.*out*.println("Шифровка простой заменой: " + cipherText);  
  
 String[] cipherText2 = "ьюинлцщё".split("");  
 sb.delete(0, sb.length());  
 int a = 8, b = 29, m = 33;  
 for(int i = 0; i < cipherText2.length; i++) {  
 int y = (a \* (alphabet2.indexOf(cipherText2[i]) + m - b)) % m;  
 sb.append(alphabet2.get(y));  
 }  
 System.*out*.println("Дешифровка аффинного шифра: " + sb.toString());  
 }  
}

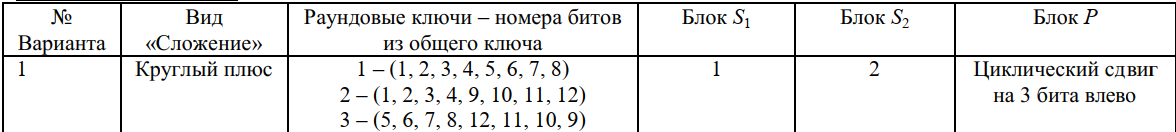
Результат

Шифровка простой заменой: шжхклёзжьщхе

Дешифровка аффинного шифра: апельсин

**Лабораторная работа 2**





Код

import java.util.List;  
import java.util.stream.Collectors;  
  
public class Main {  
 public static void main(String[] args) {  
 String[] alphabet = "0123456789abcdef".split("");  
 String[] s1 = "96328b17a4efc0d5".split("");  
 String[] s2 = "37e98af0526cb4d1".split("");  
 List<String> alph = List.*of*(alphabet);  
 int x = 0b0010\_0111;  
 int k = 0b1101\_1111\_0000;  
 int[] ki = {0b1101\_1111, 0b1101\_0000, 0b1111\_0000};  
 int y = x;  
 System.*out*.println("Результат трёх итераций SP-подстановки:");  
 for(int i = 0; i < 3; i++) {  
 int t = y^ki[i];  
 String t1 = Integer.*toHexString*(t).substring(0, 1);  
 String t2 = Integer.*toHexString*(t).substring(1, 2);  
 String n1 = s1[alph.indexOf(t1)];  
 String n2 = s2[alph.indexOf(t2)];  
 String n = String.*format*("%8s", Integer.*toBinaryString*(Integer.*parseInt*(n1 + n2, 16)))  
 .replace(' ', '0');  
 List<Character> n\_list = n.chars().mapToObj(e -> (char)e).collect(Collectors.*toList*());  
 Collections.*rotate*(n\_list, 5);  
 y = Integer.*parseInt*(n\_list.stream().map(e -> e.toString()).collect(Collectors.*joining*()), 2);  
 System.*out*.println(String.*format*("%8s", Integer.*toBinaryString*(y)).replace(' ','0'));  
 }  
 }  
}

Результат

Результат трёх итераций SP-подстановки:

10101010

10110011

01001100

**Лабораторная 3**

Для выработки ключей криптосистем и гаммы нужно использовать криптографические генераторы псевдослучайных чисел. Необходимость отказа от обычных программных генераторов (например, от линейного конгруэнтного метода) обусловлена двумя причинами: незащищенностью реализации и возможностью предугадать значение. Выделяют класс криптографических генераторов, базовым элементом которых является регистр сдвига с линейной обратной связью (РСЛОС). При этом, как правило, требуется, чтобы РСЛОС генерировал линейную рекуррентную последовательность (ЛРП) с максимально возможным периодом (относительно *длины* *РСЛОС* (количества ячеек памяти у РСЛОС)), высокой линейной сложностью и хорошими статистическими свойствами. Так как ЛРП не обладает нелинейными свойствами, то возможно восстановление РСЛОС по небольшому отрезку ЛРП этого РСЛОС. Поэтому, для построения криптографических генераторов использует сочетание РСЛОС с различными узлами и элементами памяти. Одним из вариантов такого сочетания является комбинирующий генератор. Дополнительную информацию вы можете посмотреть в лекции.

*Комбинирующий генератор.* Криптографически стойкий комбинирующий генератор построен на основе РСЛОС над полем , длины которых попарно взаимно-простые числа, и нелинейной булевой функции , на вход которой поступают элементы ЛРП, вырабатываемые РСЛОС. Таким образом, -ый элемент выходной последовательности, формируемой комбинирующим генератором, может быть вычислен как , где – -ый элемент ЛРП, вырабатываемой -ым РСЛОС. Важной характеристикой комбинирующего генератора, является линейная сложность. Высокую линейную сложность имеет, например, генератор Геффе.

*Генератор Геффе.* Этот генератор один из вариантов комбинирующего генератора при . В нем -ый элемент выходной последовательности может быть вычислен как , где – -ый элемент ЛРП, вырабатываемой -ым РСЛОС. Для достижения максимального периода выходной последовательности, длины РСЛОС должны быть попарно взаимно-простыми числами. Но даже в этом случае данный генератор уязвим к корреляционным атакам.

*Тестирование последовательностей.* Для проверки статистических свойств последовательностей, которые вырабатывают аппаратные и программные генераторы псевдослучайных чисел используются пакеты (батареи) статистических тестов. Один из самых известных пакетов – пакет тестов NIST.

*Условие лабораторной работы.* В рамках данной лабораторной работы вам необходимо реализовать генератор Геффе с заданными характеристическими многочленами и начальными состояниями.

Шаг 1. Реализовать работу РСЛОС. На входе заданы число *n* – количество ячеек памяти, из которых состоит регистр, и две битовые последовательности: 1) *an*-1*an*-2...*a*1*a*0 – задает начальное заполнение, 2) *cn*-1*cn*-2...*c*1*c*0 – задает линейную обратную связь. *a*0 – задает состояние 0-ой ячейки памяти, *a*1 – перовой и т.д., *an*-1 – (*n*-1)-ой, таким образом задаем состояния всех *n* ячеек. Пусть *b*0, *b*1, ..., *bn*-1 – текущие значения в соответствующих ячейках памяти (в начальный момент *bi* = *ai*). За одну итерацию: 1) внешний выводе: регистр возвращает значение, хранящееся в 0-ой ячейке, т.е. *b*0; 2) изменение внутреннего состояния: вычисляется значение *r* = (*c*0\**b*0) + (*c*1\**b*1) +...+ (*cn*-1\**bn*-1); значение *i*-ой ячейки переходит в (*i*-1)-ую (*i* изменяется от 1 до *n*-1), т.е. *bi*-1 = *bi*; значение (*n*-1)-ой принимается равным *bn*-1 = *r*. На рисунке внизу представлен пример, для n = 8 и *c7c6*...*c*1*c*0=00011101.

На первом шаге необходимо реализовать 3 РСЛОС, заданные в вашем варианте. Для каждого регистра найти период выходной последовательности и сгенерированную последовательность, до начала зацикливания.

Шаг 2. Сгенерировать выходную последовательность *γi* генератора Геффе длительностью N = 10 000 элементов.

Шаг 3. Для сгенерированной на Шаге 2 последовательности вычислить следующие статистики:  
1) количество 0 и количество 1; 2)  для *i* от 1 до 5. Прокомментировать полученные результаты.

|  |  |  |  |
| --- | --- | --- | --- |
| **Вариант** | **РСЛОС №1**  **(5)** | **РСЛОС №2**  **(7)** | **РСЛОС №3**  **(8)** |
| 1 | 01101  10111 | 1101111  1100001 | 00000011  11001101 |

Код

import java.util.\*;

public class Main {

public static void main(String[] args) {

LinkedList<Boolean> a1 = new LinkedList<>(Arrays.asList(false, true, true, false, true));

LinkedList<Boolean> a2 = new LinkedList<>(Arrays.asList(true, true, false, true, true, true, true));

LinkedList<Boolean> a3 = new LinkedList<>(Arrays.asList(false, false, false, false, false, false, true, true));

List<Boolean> c1 = Arrays.asList(true, false, true, true, true);

List<Boolean> c2 = Arrays.asList(true, true, false, false, false, false, true);

List<Boolean> c3 = Arrays.asList(true, true, false, false, true, true, false, true);

LinkedList<Boolean> s1 = LFSR(a1, c1);

LinkedList<Boolean> s2 = LFSR(a2, c2);

LinkedList<Boolean> s3 = LFSR(a3, c3);

List<Boolean> y = geffe(s1, s2, s3);

System.out.println();

System.out.println("y: " + y);

long one\_count = y.stream().filter(e -> e).count();

long zero\_count = 10\_000l - one\_count;

System.out.println("Нули: " + zero\_count);

System.out.println("Единицы: " + one\_count);

for(int i = 1; i <= 5; i++) {

int r = 0;

for(int j = 0; j < 10\_000-i; j++) {

int x = y.get(j) ^ y.get(j+1) ? 1 : 0;

int tau = (int) Math.pow(-1, x);

r += tau;

}

System.out.println("r\_" + i + ": " + r);

}

}

static List<Boolean> geffe(LinkedList<Boolean> s1, LinkedList<Boolean> s2, LinkedList<Boolean> s3) {

List<Boolean> y = new ArrayList<>();

ArrayList<Boolean> s1\_new = new ArrayList<>(s1);

ArrayList<Boolean> s2\_new = new ArrayList<>(s2);

ArrayList<Boolean> s3\_new = new ArrayList<>(s3);

for(int i = 0; i < 10\_000; i++) {

Boolean y\_t = (s1\_new.get(i) & s2\_new.get(i)) ^ (!s1\_new.get(i) & s3\_new.get(i));

y.add(y\_t);

}

return y;

}

static LinkedList<Boolean> LFSR(LinkedList<Boolean> a, List<Boolean> c) {

LinkedList<Boolean> s = new LinkedList<>();

boolean check = true;

Set<LinkedList<Boolean>> period = new HashSet<>();

period.add(a);

int i = 0;

for(int t = 1; i < 10\_000; i++, t++) {

boolean r = false;

for(int j = 0; j < a.size(); j++)

r ^= a.get(j)&c.get(j);

s.add(a.pollLast());

a.addFirst(r);

if(check && period.contains(a)) {

System.out.println("period: " + t);

System.out.println("s: " + s);

break;

}

}

for(; i < 10\_000; i++) {

boolean r = false;

for(int j = 0; j < a.size(); j++)

r ^= a.get(j)&c.get(j);

s.add(a.pollLast());

a.addFirst(r);

}

return s;

}

}

Результат

period: 31

s: [true, false, true, true, false, false, false, false, true, true, true, false, false, true, true, false, true, true, true, true, true, false, true, false, false, false, true, false, false, true, false]

period: 60

s: [true, true, true, true, false, true, true, true, true, true, true, false, false, true, false, false, true, false, true, true, true, false, true, false, true, false, false, true, true, true, false, false, false, false, true, false, false, false, false, false, false, true, true, false, true, true, false, true, false, false, false, true, false, true, false, true, true, false, false, false]

period: 28

s: [true, true, false, false, false, false, false, false, true, false, true, true, false, false, true, false, false, true, false, true, false, true, true, true, true, false, false, true]

Нули: 5162

Единицы: 4838

r\_1: -59

r\_2: -60

r\_3: -59

r\_4: -58

r\_5: -59

**Лабораторная 4**

Формально, как и для любой ассиметричной криптосистемы, для работы RSA необходимо реализовать 3 основные процедуры: генерацию ключей, функцию зашифрования и функцию расшифрования.

Генерация ключей:

1. Выбираются два больших простых числа *p* и *q*.

2. Вычисляется число *n* = *pq*; *n* – открытый параметр.

3. Открытый ключ *e* выбирается случайным образом из Zϕ(*n*) = {0, ..., ϕ(*n*)-1} таким образом, чтобы НОД(*e*, ϕ(*n*)) = 1. (Зная *p* и *q,* можно легко вычислить ϕ(*n*) = (*p*-1)(*q*-1)).

4. Закрытый ключ *d* из Zϕ(*n*); находится по формуле *d* = *e*-1 (mod ϕ(*n*)).

Замечание: Проверить, что случайно выбранное число *e* удовлетворяет условию НОД(*e*, ϕ(*n*)) = 1, и, если условие выполнено, найти закрытый ключ *d* = *e*-1 (mod ϕ(*n*)) можно с помощью расширенного алгоритма Евклида.

Сообщение *X*, которое необходимо зашифровать, представляется в виде некоторого числа из Z*n*.

Функция зашифрования: E*e*(*X*) = *Xe* (mod *n*), где *n* – открытый параметр, *e* – открытый ключ.

Замечание: для возведения в степень рекомендуется использовать алгоритм быстрого возведения в степень. Вычислять остаток от деления необходимо после каждой операции умножения.

Функция расшифрования: D*d*(*Y*) = *Yd* (mod *n*), где *n* – открытый параметр, *d* – закрытый ключ.

Что необходимо сделать:

0) Для выполнения всех заданий необходимо использовать "длинную" арифметику. Разрешается использовать любую библиотеку для этих целей или написать свою.

1) В условиях своего варианта для заданных чисел *p*, *q* и *e*, необходимо вычислить *d*. **Важно**: использовать готовую реализацию расширенного алгоритма Евклида или вычисления обратного по модулю элемента нельзя! (Можно только для проверки своих результатов).

2) Для заданного сообщения *X*1, вычислить зашифрованное сообщение *Y*1, используя открытый ключ *e*.

3) Расшифровать сообщение *Y*1, использую закрытый ключ *d*, сравнить результат с исходным сообщением *X*1.

4) Для заданного шифртекста *Y*2, вычислить исходный открытый текст *X*2, используя закрытый ключ *d*.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № Вар. | *p* | | *q* | *e* |
| 1 | 563036103490583 | | 1063300642915937 | 372585779765210097553647509959 |
| *X*1 | | *Y*2 | | |
| 399754188907643924420059310699 | | 293314580135454643114146935352 | | |

Код

def euclid\_ext(a, b):

if a == 0:

return b, 0, 1

gcd, x, y = euclid\_ext(b % a, a)

return gcd, y - (b // a) \* x, x

def mod\_inverse(a, n):

# returns solution of a \* x = 1 (mod n)

g, x, \_ = euclid\_ext(a, n)

if g == 1:

return x % n

return None

class RSA:

def \_\_init\_\_(self, p, q, e):

self.n = p \* q

self.e = e

self.d = self.\_generate\_private\_key(p, q)

def \_generate\_private\_key(self, p, q):

phi = (p - 1) \* (q - 1)

return mod\_inverse(self.e, phi)

def encrypt(self, message):

return pow(message, self.e, self.n)

def decrypt(self, message):

return pow(message, self.d, self.n)

p = 563036103490583

q = 1063300642915937

e = 372585779765210097553647509959

X1 = 399754188907643924420059310699

Y2 = 293314580135454643114146935352

rsa = RSA(p, q, e)

print(f'Заданное сообщение X1: {X1}')

Y1 = rsa.encrypt(X1)

print(f'Зашифрованное сообщение Y1: {Y1}')

X1\_dec = rsa.decrypt(Y1)

print(f'Расшифрованное сообщение: {X1\_dec}')

print()

print(f'Зашифрованное сообщение Y2: {Y2}')

X2\_dec = rsa.decrypt(Y2)

print(f'Расшифрованное сообщение: {X2\_dec}')

Результат

Заданное сообщение X1: 399754188907643924420059310699

Зашифрованное сообщение Y1: 538724514668280580618317050875

Расшифрованное сообщение: 399754188907643924420059310699

Зашифрованное сообщение Y2: 293314580135454643114146935352

Расшифрованное сообщение: 35891091969105790935824030377

**Лабораторная работа 5**

Формально, как и для любой схемы электронно-цифровой подписи (ЭЦП), для работы схемы Эль-Гамаля необходимо реализовать 3 основных алгоритма: генерацию ключей, функцию подписи и функцию проверки подписи. Подробности можно посмотреть в лекции. С учетом того, что в лабораторной не требуется реализовывать генерацию простых чисел, алгоритмы ЭЦП имеют следующий вид:

*Генерация ключей (Gen)*

*Вход: q –* простое число. *Шаги:*

1. Выбрать такое четное число *R,* что *R < 4 (q + 1).*

2. Вычислить число *p = qR + 1*.

3. Если *2qR ≠ 1 (mod p)* или *2R = 1 (mod p),* то возвратиться к шагу 1, иначе *p* – простое.

4. Случайным образом выбрать *x* из *Zp* и вычислить *g = xR (mod p).*

5. Если *g = 1*, то возвратиться к шагу 4, иначе искомое *g* найдено.

6. Случайным образом выбрать личный ключ *d* из *Zq.*

7. Вычислить открытый ключ *e = gd (mod p).*

*Выход: (p, q, g)* – параметры ЭЦП; e – открытый ключ; *d* – личный ключ.

*Функция подписи (Sign)*

*Вход: (p, q, g)* – параметры ЭЦП; *d* – личный ключ; *M* – подписываемое сообщение (в виде строки текста произвольной длины).

*Шаги:*

1. Вычислить хэш-значение *m* от сообщения *M*: *m = h(M)* (*h()* – хэш-функция).

2. Случайным образом выбрать одноразовый личный ключ *k* из *Zq / {0}.*

3. Вычислить *r = gk (mod p)*.

4. Вычислить *s = k-1(m – dr) (mod q). Выход: (r, s)* – подпись.

*Функция проверки подписи (Verify)*

*Вход: (p, q, g)* – параметры ЭЦП; *e* – открытый ключ; *M* – подписываемое сообщение (в виде строки текста произвольной длины; *(r, s)* – подпись.

*Шаги:*

1. Если *r* не лежит в *Zp / {0}* или *s* не лежит в *Zq*, то вернуть *FALSE.*

2. Вычислить хэш-значение *m* от сообщения *M*: *m = h(M)* (*h()* – хэш-функция).

3. Если *errs = gm (mod p),* то вернуть *TRUE*, иначе вернуть *FALSE. Выход: TRUE,* если подпись корректна; *FALSE,* если подпись некорректна.

*Условие лабораторной работы:*

1) Для выполнения всех заданий необходимо использовать "длинную" арифметику. Разрешается использовать любую готовую библиотеку или написать свою. При этом учитывайте, что допустимые языки программирования {С/С++, C#, Java, Python, R}.

*2)* Ваша программа должна принимать на вход указание, какую операцию необходимо выполнить: *Gen*, *Sign*, *Verify или Exit*. В зависимости от операции программа должна считывать (из консоли, файла и т.п.) необходимые данные, выполнять шаги выбранного алгоритма и возвращать (в консоль, файл и т.п.) результат выполнения соответствующей операции*.* При вызове операции *Exit* программа не считывает входных данных, а просто завершает работу.

3) Программа должна иметь проверку корректности входных данных (*d* лежит в *Zq* для *Sign*; *e* лежит в *Zp* для *Verify*).

4) Программа должна предлагать пользователю выполнить очередную операцию до тех пор, пока пользователь не выберет вариант выхода из программы.

|  |  |
| --- | --- |
| **№ вар.** | **q** |
| 1 | 140990220132661942094353836270106675983065715626747592141413983992936990059339 |

Код

from random import randrange

from hashlib import sha256

def euclid\_ext(a, b):

if a == 0:

return b, 0, 1

gcd, x, y = euclid\_ext(b % a, a)

return gcd, y - (b // a) \* x, x

def mod\_inverse(a, n):

# returns solution of a \* x = 1 (mod n)

g, x, \_ = euclid\_ext(a, n)

if g == 1:

return x % n

return None

def pow\_(x, n, mod):

if n == 0:

return 1

if n % 2 == 0:

return pow\_(((x % mod) \* (x % mod)) % mod, n // 2, mod)

else:

return ((x % mod) \* pow\_((x % mod), n - 1, mod)) % mod

def gen(q):

while True:

r = randrange(0, 4 \* (q + 1), 2)

p = q \* r + 1

if pow\_(2, q \* r, p) != 1 or pow\_(2, r, p) == 1:

continue

while True:

x = randrange(0, p)

g = pow\_(x, r, p)

if g == 1:

continue

else:

break

d = randrange(0, q)

e = pow\_(g, d, p)

break

DS\_params = (p, q, g)

open\_key = e

personal\_key = d

return DS\_params, open\_key, personal\_key

def sign(signature\_params, personal\_key, message):

p, q, g = signature\_params

d = personal\_key

if not (0 <= personal\_key <= q):

raise ValueError()

m = int(sha256(bytes(message, encoding="utf-8")).hexdigest(), 16)

k = randrange(1, q)

r = pow\_(g, k, p)

s = (mod\_inverse(k, q) \* (m - d \* r)) % q

return r, s

def verify(signature\_params, open\_key, message, signature):

p, q, g = signature\_params

r, s = signature

e = open\_key

if not (0 <= open\_key < p):

raise ValueError()

m = int(sha256(bytes(message, encoding="utf-8")).hexdigest(), 16)

if (pow\_(e, r, p) \* pow\_(r, s, p)) % p == pow\_(g, m, p):

return True

return False

def main():

signature\_params, open\_key, personal\_key, r, s, message = None, None, None, None, None, None

while True:

operation = input(

"Enter gen for keys signature, sign for signing message, verify to verify signature, "

+ "exit to exit the program: "

)

if operation == "exit":

return

if operation == "gen":

q = int(input("Enter q: "))

signature\_params, open\_key, personal\_key = gen(q)

print(f"open key = {open\_key}")

print(f"personal key = {personal\_key}")

continue

if operation == "sign":

if signature\_params is None:

print("Keys need to be generated first, use gen command")

continue

else:

message = input("Enter message:")

r, s = sign(signature\_params, personal\_key, message)

print(f"r = {r}")

print(f"s = {s}")

continue

if operation == "verify":

if signature\_params is None:

print("Keys need to be generated first, use gen command")

continue

else:

print(verify(signature\_params, open\_key, message, (r, s)))

continue

main()

Результат

Enter gen for keys signature, sign for signing message, verify to verify signature, exit to exit the program: gen

Enter q: 140990220132661942094353836270106675983065715626747592141413983992936990059339

open key = 11780877586853349277333943333075659577444361004095809819988749417709864712942503544745945009780184512344180874692603022521384412295090122921849107108049825

personal key = 2821762382840998541904572037119953876443039782147645144472155896762147247605

Enter gen for keys signature, sign for signing message, verify to verify signature, exit to exit the program: sign

Enter message:music

r = 227293879071965218660524100870796990309432576914715194521956845095066584696094502155373212281781346017577285299042624188332790111158994960949945082410410

s = 89865847165411352334427921885385388717988255429096050031620761049055637528464

Enter gen for keys signature, sign for signing message, verify to verify signature, exit to exit the program: verify

True

Enter gen for keys signature, sign for signing message, verify to verify signature, exit to exit the program: exit