МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ

образовательное учреждение

высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра защиты информации

**

**Практическая работа №2**

**«**Изучение математических основ криптографии**»**

**по дисциплине: «*Программирование*»**

|  |  |
| --- | --- |
| Выполнил:  Студент гр. «АБc-323», «АВТФ»  *Раков Михаил Андреевич*  «15» мая 2024г  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | Проверил:  *Доцент кафедры ЗИ*  *Архипова Анастасия Борисовна*  «\_\_\_» \_\_\_\_\_\_ 2024г  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |

Новосибирск 2024

**Цели и задачи работы**: изучение циклических алгоритмов, операторов цикла, программирование циклического вычислительного процесса на примере математических методов основ криптографии.

**Задание к работе**: реализовать циклический вычислительный процесс. Самостоятельно решить задачи в соответствии с индивидуальным вариантом.

**Задание 1**. Реализовать a^x mod p Сравнения по модулю простого числа через теорему Ферма и свойства сравнений. Программа должна проверять условия выполнения теоремы Ферма и простоту вводимого пользователем модуля. Реализовать алгоритм через разложение степени в двоичный вид (логарифм)^2.

**Задание 2**. Реализовать обобщенный алгоритм Евклида для вычисления с\*d mod m=1. Реализовать расширенный алгоритм Евклида для вычисления взаимообратного числа с^(-1) mod m=d.

**Задание 3**. Написать программу, использующую алгоритм шифрования данных для преобразования исходного текста (схема разделения ключа Шамира).

**Задание 4**. Найти последнюю цифру «трехэтажного числа». Например, 3^(7^8).

**Задание 5**. Написать сообщение на тему «Атака посередине».

**Задание 6**. Написать сообщение на тему «Стандарты современной криптографии в РФ».

# **Задания 1, 2, 4**

**Функция Эйлера** φ(n) определяется как количество натуральных чисел, не превосходящих n и взаимно простых с n.

**Теорема Эйлера** – это теорема теории чисел, которая утверждает, что если a и m взаимно просты, то a^ φ(n) делится на m с остатком 1.



**Малая теорема Ферма** — это теорема теории чисел, которая утверждает, что если p-простое число, а-целое число, то а^(p-1)-1 делится на p с остатком 1.



Листинг программы на C++:

Файл main.cpp

#include <iostream>

#include "modular\_arithmetic.h"

using namespace std;

int main()

{

num\_type a, x, p, b;

cout << "Сравнение числа a в степени x "

<< "по модулю простого числа p." << endl

<< "Введите a, x и p: " << endl;

cin >> a >> x >> p;

cout << "Алгоритм, основанный на теореме Ферма: ";

try

{

cout << pow\_mod\_Fermats(a, x, p) << endl;

}

catch (const invalid\_argument& exc)

{

cout << exc.what() << endl;

}

cout << "Через разложение степени: " << pow\_mod(a, x, p) << endl;

cout << "\nРасширенный алгоритм Евклида." << endl

<< "Введите a и b: ";

cin >> a >> b;

auto [v, u, d] = nod\_extended(a, b, true);

cout << "НОД(a, b) = " << d << endl

<< "v\*a + u\*b = d: "

<< v << "\*" << a << " + " << u << "\*" << b

<< " = " << d << endl;

cout << "\nЧисло обратное c по модулю m" << endl

<< "Введите c и m: ";

cin >> a >> p;

try

{

cout << "Обратное " << a << " по модулю " << p

<< " = " << inverse\_mod(a, p) << endl;

}

catch (const invalid\_argument& exc)

{

cout << exc.what() << endl;

}

cout << "\nПоследняя цифра трёхэтажного числа: ";

cin >> a >> b >> p;

cout << last\_digit(a, b, p) << endl;

}

Файл modular\_arithmetic.cpp

#include <iostream>

#include "modular\_arithmetic.h"

using namespace std;

num\_type mod (num\_type a, num\_type b)

{

return a >= 0 ? a % b

: b + (a % b);

}

bool is\_prime (num\_type n)

{

if (n == 2)

return true;

if (n < 2 or mod(n, 2) == 0)

return false;

for (num\_type i = 3; i <= sqrt(n); ++i)

{

if (mod(n, i) == 0)

return false;

}

return true;

}

num\_type pow\_mod\_Fermats (num\_type a, num\_type x, num\_type p)

{

if (mod(a, p) == 0)

throw invalid\_argument("ошибка: a делиться на p");

if (not is\_prime(p))

throw invalid\_argument("ошибка: p - составное число");

num\_type result = 1;

x = mod(x, p - 1);

for (num\_type i = 0; i < x; ++i)

{

result = mod(result \* a, p);

}

return result;

}

num\_type pow\_mod(num\_type a, num\_type x, num\_type p)

{

num\_type result = 1;

while (x > 0)

{

if (mod(x, 2) == 1)

result = mod(result \* a, p);

a = mod(a \* a, p);

x /= 2;

}

return result;

}

num\_type nod (num\_type a, num\_type b)

{

while (a % b != 0)

{

num\_type oldB = b;

b = a % b;

a = oldB;

}

return b;

}

tuple<num\_type, num\_type, num\_type>

nod\_extended (num\_type a, num\_type b, bool print)

{

num\_type vPrev = 1, uPrev = 0;

num\_type v = 0, u = 1;

num\_type r;

while ((r = mod(a, b)) != 0)

{

num\_type tV = vPrev;

num\_type tU = uPrev;

vPrev = v;

uPrev = u;

v = tV - (a / b) \* vPrev;

u = tU - (a / b) \* uPrev;

if (print)

cout << a << " " << v << " " << b << " " << u

<< " " << r << endl;

a = b, b = r;

}

if (print)

cout << a << " " << v << " " << b << " " << u

<< " " << r << endl;

return make\_tuple(v, u, b);

}

num\_type inverse\_mod (num\_type a, num\_type m)

{

auto [v, u, d] = nod\_extended(a, m, false);

if (d != 1)

throw invalid\_argument("ошибка, числа a и m должны быть взаимно просты");

return mod(v, m);

}

num\_type func\_Euler (num\_type m)

{

num\_type cnt = 0;

for (num\_type i = 2; i < m; ++i)

{

if (nod(m, i) == 1)

++cnt;

}

return cnt;

}

num\_type last\_digit (num\_type a, num\_type b, num\_type c)

{

num\_type pwr = pow\_mod(b, c, func\_Euler(10));

return pow\_mod(a, pwr, 10);

}

Файл modular\_arithmetic.h

#ifndef MODULAR\_ARITHMETIC\_H

#define MODULAR\_ARITHMETIC\_H

#include <cmath>

#include <tuple>

#include <stdexcept>

using num\_type = long long;

num\_type mod (num\_type a, num\_type b);

bool is\_prime (num\_type);

num\_type pow\_mod\_Fermats (num\_type a, num\_type x, num\_type p);

num\_type pow\_mod(num\_type a, num\_type x, num\_type p);

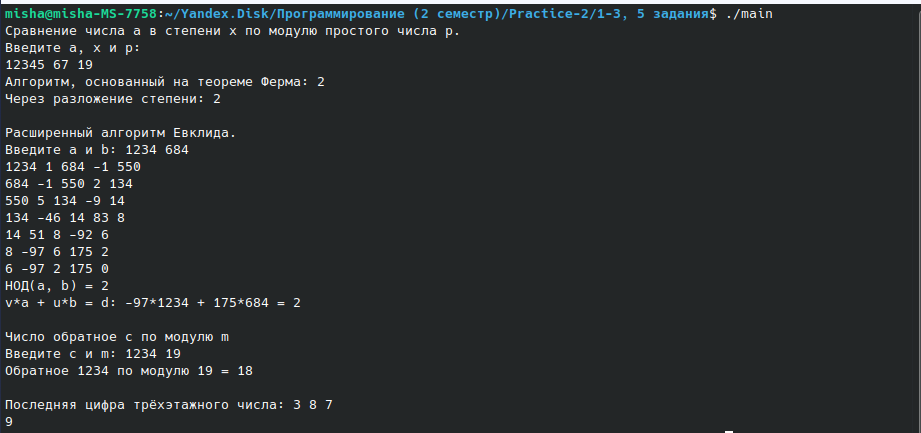
std::tuple<num\_type, num\_type, num\_type>

nod\_extended (num\_type a, num\_type b, bool);

num\_type inverse\_mod (num\_type a, num\_type m);

num\_type last\_digit (num\_type a, num\_type b, num\_type c);

#endif

Результаты работы программы:  


Листинг программы на Haskell:

(?) cond a = if cond then (\x -> a) else (\x -> x)

-- проверка на простоту

isPrime :: Int -> Bool

isPrime n

| n == 2 = True

| n < 2 || even n = False

| otherwise = all (\i -> n `mod` i /= 0) [3 .. n-1]

powModF :: Int -> Int -> Int -> Int

powModF a x p

| a `mod` p == 0 = error "ошибка: a делиться на p"

| not $ isPrime p = error "ошибка: p - составное число"

| otherwise = foldl (\r nA -> r \* nA `mod` p) 1 repA

where newX = x `mod` (p-1)

repA = take newX $ repeat a

-- возведенение в степень по модулю

powMod :: Int -> Int -> Int -> Int

powMod \_ 0 \_ = 1

powMod a power n = if even power

then result

else result \* a `mod` n

where newA = a \* a `mod` n

newPower = power `div` 2

result = powMod newA newPower n

nod :: Int -> Int -> Int

nod r 0 = r

nod a b = nod b (a `mod` b)

nodExtend' :: Int -> Int -> [Int] -> [Int] -> (Int, Int, Int)

nodExtend' a b [v, vP] [u, uP] =

if a `mod` b == 0

then (v, u, b)

else nodExtend' b r [newV, v] [newU, u]

where (d, r) = divMod a b

newV = vP - d \* v

newU = uP - d \* u

nodExtend :: Int -> Int -> (Int, Int, Int)

nodExtend a b = nodExtend' a b [0, 1] [1, 0]

inverseMod :: Int -> Int -> Int

inverseMod a b =

if d /= 1

then error"ошибка, числа a и m должны быть взаимно просты"

else v `mod` m

where (v, u, d) = nodExtend a b

funcEuler :: Int -> Int

funcEuler m = length $ filter f [2 .. m-1]

where f = (\i -> m `nod` i == 1)

lastDigit Int -> Int -> Int -> Int

lastDigit a b c = powMod a x 10

where x = powMod b c $ funcEuler 10

# **Задание 3**

Схема разделения ключа Шамира – это, так называемая, (n, k) пороговая схема для эффективного разделения секретного значения (например, криптографического ключа) на n частей. Затем, когда и только когда хотя бы k из n частей собраны, можно легко восстановить секрет S.

Пороговая схема Шамира (n, k) построена вокруг концепции полиномиальной интерполяции.

Листинг программы на C++:

Файл modular\_arithmetic.cpp

#include "modular\_arithmetic.h"

using namespace std;

num\_type mod (num\_type a, num\_type b)

{

return a >= 0 ? a % b

: b + (a % b);

}

bool is\_prime (num\_type n)

{

if (n == 2)

return true;

if (n < 2 or mod(n, 2) == 0)

return false;

for (num\_type i = 3; i <= sqrt(n); ++i)

{

if (mod(n, i) == 0)

return false;

}

return true;

}

num\_type pow\_mod\_Fermats (num\_type a, num\_type x, num\_type p)

{

if (mod(a, p) == 0)

throw invalid\_argument("ошибка: a делиться на p");

if (not is\_prime(p))

throw invalid\_argument("ошибка: p - составное число");

num\_type result = 1;

x = mod(x, p - 1);

for (num\_type i = 0; i < x; ++i)

{

result = mod(result \* a, p);

}

return result;

}

num\_type pow\_mod(num\_type a, num\_type x, num\_type p)

{

num\_type result = 1;

while (x > 0)

{

if (mod(x, 2) == 1)

result = mod(result \* a, p);

a = mod(a \* a, p);

x /= 2;

}

return result;

}

tuple<num\_type, num\_type, num\_type>

nod\_extended (num\_type a, num\_type b)

{

num\_type vPrev = 1, uPrev = 0;

num\_type v = 0, u = 1;

num\_type r;

while ((r = mod(a, b)) != 0)

{

num\_type tV = vPrev;

num\_type tU = uPrev;

vPrev = v;

uPrev = u;

v = tV - (a / b) \* vPrev;

u = tU - (a / b) \* uPrev;

a = b, b = r;

}

return make\_tuple(v, u, b);

}

Файл modular\_arithmetic.h

#ifndef MODULAR\_ARITHMETIC\_H

#define MODULAR\_ARITHMETIC\_H

#include <cmath>

#include <tuple>

#include <stdexcept>

using num\_type = long long;

num\_type mod (num\_type a, num\_type b);

bool is\_prime (num\_type);

num\_type pow\_mod\_Fermats (num\_type a, num\_type x, num\_type p);

num\_type pow\_mod(num\_type a, num\_type x, num\_type p);

std::tuple<num\_type, num\_type, num\_type>

nod\_extended (num\_type a, num\_type b);

#endif

Файл shamir\_scheme.cpp

#include "shamirs\_scheme.h"

using namespace std;

vector<num\_type> rand\_unique\_natural\_nums (int k)

{

set<num\_type> nums;

while (nums.size() < k)

{

nums.insert(rand() + 1);

}

return vector<num\_type> (nums.begin(), nums.end());

}

vector<key\_type>

keys\_generation (num\_type secret, int n, int k, num\_type m)

{

vector<num\_type> xs = rand\_unique\_natural\_nums (n);

vector<num\_type> coeffs = rand\_unique\_natural\_nums (k - 1);

coeffs.insert(coeffs.begin(), secret);

vector<key\_type> keys;

for (auto x : xs)

{

num\_type y = 0;

for (size\_t i = 0; i < coeffs.size(); ++i)

{

num\_type mono = coeffs[i] \* pow\_mod(x, i, m);

y = (y + mono) % m;

}

keys.push\_back( {x, y} );

}

return keys;

}

num\_type secret\_recovery (const vector<key\_type>& keys, num\_type p)

{

num\_type secret = 0;

for (auto [xJ, yJ] : keys)

{

num\_type num = yJ;

num\_type denom = 1;

for (auto [x, ignore\_y] : keys)

{

if (x == xJ)

continue;

num = mod(num \* -x, p);

denom = denom \* pow\_mod(xJ - x, p - 2, p);

}

secret = mod(secret + num \* denom, p);

}

return secret;

}

pair <char, num\_type>

pseudo\_rand\_symbol (num\_type x\_n, num\_type M)

{

char rndS = 0;

for (int i = 0; i < 8; ++i)

{

x\_n = pow\_mod (x\_n, 2, M);

rndS = rndS \* 2 + (x\_n % 2);

}

return {rndS, x\_n};

}

void encode\_and\_decode (string& s, num\_type secret\_key, num\_type M)

{

num\_type x\_n = secret\_key;

for (auto& ch : s)

{

char rndS;

tie(rndS, x\_n) = pseudo\_rand\_symbol(x\_n, M);

ch = ch ^ rndS;

}

}

Файл shamir\_scheme.h

#ifndef SHAMIR\_SCHEME\_H

#define SHAMIR\_SCHEME\_H

#include <vector>

#include <cmath>

#include <set>

#include "modular\_arithmetic.h"

using key\_type = std::pair<num\_type, num\_type>;

// генерирует n различных ключей,

// k из которых необходимо для восстановления секрета

std::vector<key\_type>

keys\_generation (num\_type secret, int n, int k, num\_type m);

// восстанавливает секрет по переданным ключам

num\_type secret\_recovery (const std::vector<key\_type>& points, num\_type m);

// кодирует и декодирует строку, используя секретный ключ

void encode\_and\_decode (std::string& s, num\_type secret\_key, num\_type M);

#endif

Файл main.cpp

#include <iostream>

#include <fstream>

#include <string>

#include "shamirs\_scheme.h"

using namespace std;

string file\_to\_string (const string& name)

{

string res, s;

ifstream in (name);

while (getline(in, s))

{

res += (s + "\n");

}

in.close();

return res;

}

int main()

{

num\_type prime = 3701;

num\_type p = 6025631;

num\_type q = 8478859;

num\_type secret = rand() % prime;

int n, k;

cout << "Введите n и k: ";

cin >> n >> k;

string fname;

cout << "Введите имя файла: ";

cin >> fname;

vector<key\_type> gen\_keys = keys\_generation(secret, n, k, prime);

string text = file\_to\_string(fname);

encode\_and\_decode(text, secret, p \* q);

ofstream encode ("encode.txt");

encode << text;

encode.close();

cout << "\nКлючи:" << endl;

for (auto [x, y] : gen\_keys)

{

cout << x << " " << y << endl;

}

cout << "\nВведите " << k << " ключа" << endl;

vector<key\_type> keys(k);

for (auto& [x, y] : keys)

{

cin >> x >> y;

}

secret = secret\_recovery (keys, prime);

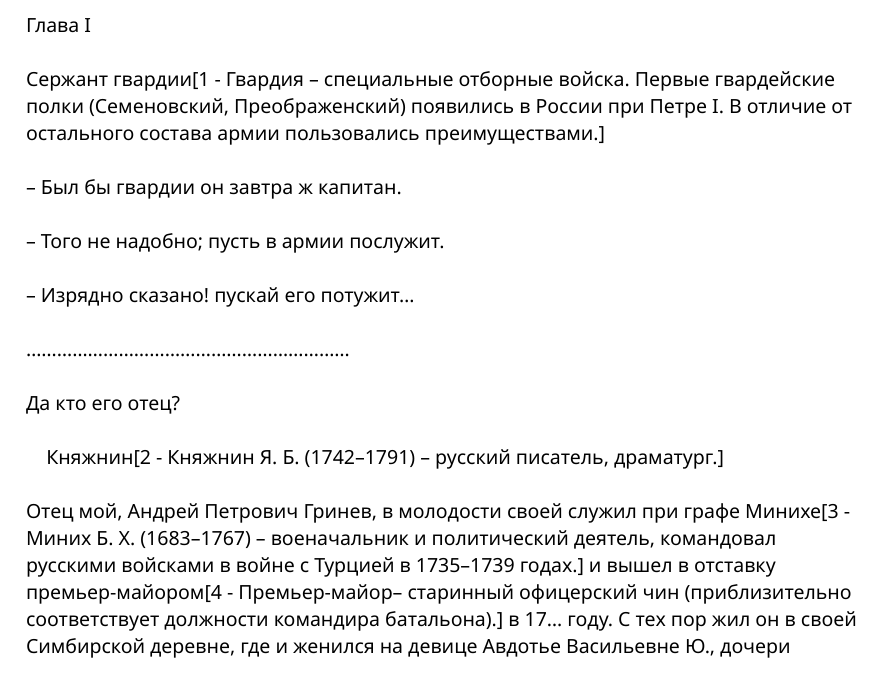
encode\_and\_decode(text, secret, p \* q);

ofstream decode ("decode.txt");

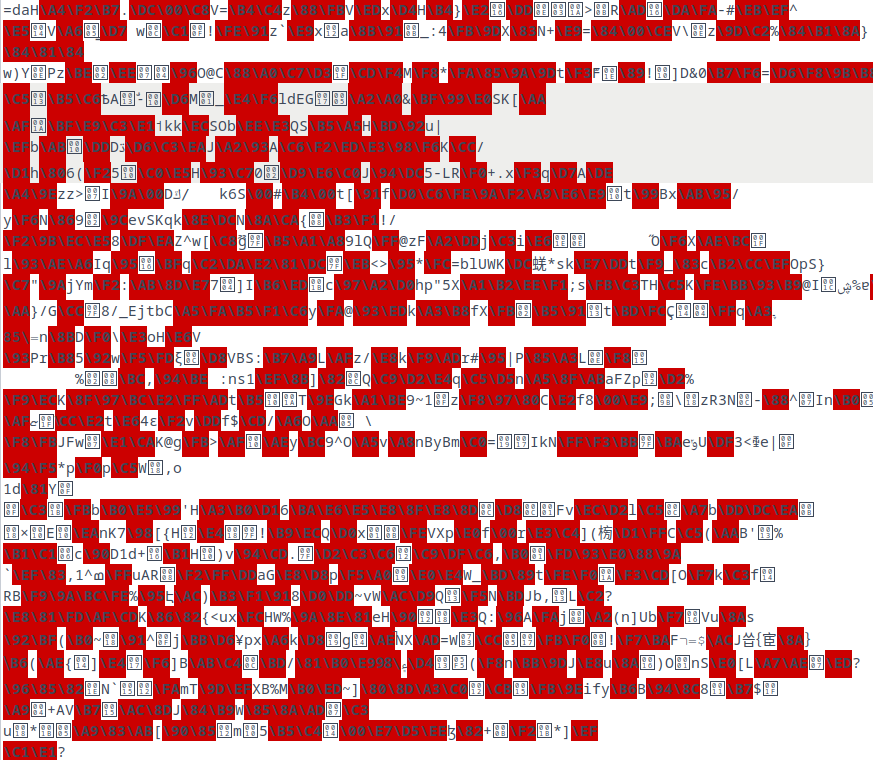
decode << text;

decode.close();

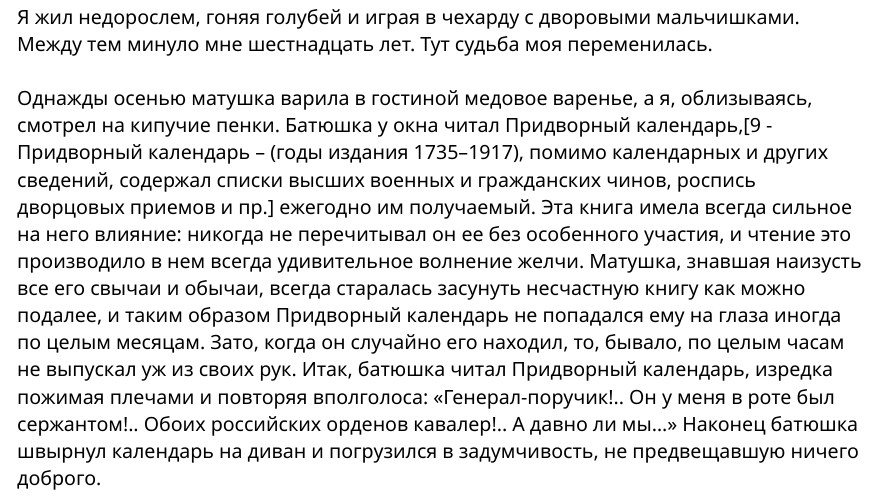
}

Результат работы программы:  


Фрагмент текста книги А.С. Пушкина «Капитанская дочка».



Фрагмент зашифрованного текста книги А.С. Пушкина «Капитанская дочка».



Дешифрованный фрагмент текста.

Листинг программы на Haskell:

import System.IO

import System.Random

import Data.Set (fromList, toList)

powMod :: (Integral a) => a -> a -> a -> a

powMod \_ 0 \_ = 1

powMod a power n = if even power

then result

else result \* a `mod` n

where newA = a \* a `mod` n

newPower = power `div` 2

result = powMod newA newPower n

class (Integral a) => FModulo a where

modulo :: a

toFModulo :: a -> a

toFModulo a = a `mod` modulo

addM :: a -> a -> a

addM a b = toFModulo (toFModulo a + toFModulo b)

subM :: a -> a -> a

subM a b = toFModulo (toFModulo a - toFModulo b)

mulM :: a -> a -> a

mulM a b = toFModulo (toFModulo a \* toFModulo b)

powM :: a -> a -> a

powM \_ 0 = 1

powM a n = if even n

then result

else result `mulM` a

where newN = n `div` 2

result = (a `mulM` a) `powM` newN

divM :: a -> a -> a

divM a b = a `mulM` invB

where invB = b `powM` (modulo-2)

instance FModulo Int where

modulo = 3701

polinomeM :: [Int] -> Int -> Int

polinomeM [] \_ = 0

polinomeM (c:coefs) x = c `addM` n\_1Nome

where n\_1Nome = x `mulM` polinomeM coefs x

type Key = (Int, Int)

keysGen' :: [Int] -> [Int] -> [Key]

keysGen' coefs xs = zip xs ys

where ys = map (polinomeM coefs) xs

keysGen :: Int -> Int -> Int -> StdGen -> [Key]

keysGen secret n k gen = keysGen' coefs xs

where coefs' = randUniqueNums (k-1) gen

coefs = secret : map toFModulo coefs'

xs = randUniqueNums n $ snd $ next gen

monomeLagrange :: Int -> Int -> [Int] -> Int

monomeLagrange yJ xJ xs' = num `mulM` denom

where xs = filter (/= xJ) xs'

num = foldl mulM yJ $ map negate xs

denom = foldl divM 1 $ map (xJ -) xs

secretRecovery :: [Key] -> Int

secretRecovery keys = foldl addM 0 monomes

where xs = map fst keys

monomeF = (\(x, y) -> monomeLagrange y x xs)

monomes = map monomeF keys

rand :: IO Int

rand = randomIO

randUniqueNums :: Int -> StdGen -> [Int]

randUniqueNums n gen = take n newL

where rndL = take (n\*10) $ (randoms gen :: [Int])

newL = toList $ fromList rndL

data Bit = B0 | B1 deriving (Eq, Ord, Enum)

type Bits = [Bit]

intToBits' :: Int -> Bits

intToBits' 0 = [B0]

intToBits' 1 = [B1]

intToBits' n = lastBit : intToBits' (n `div` 2)

where lastBit = if even n then B0 else B1

maxBits :: Int

maxBits = length $ intToBits' $ fromEnum (maxBound :: Char)

intToBits :: Int -> Bits

intToBits n = remBits ++ bits

where bits = reverse (intToBits' n)

lenRem = maxBits - (length bits)

remBits = take lenRem $ cycle [B0]

charToBits :: Char -> Bits

charToBits char = intToBits (fromEnum char)

bitsToInt :: Bits -> Int

bitsToInt bits = sum $ zipWith powF bits pows

where size = length bits

pows = reverse $ take size [0 ..]

powF = (\b p -> if b == B1 then 2 ^ p else 0)

bitsToChar :: Bits -> Char

bitsToChar bits = toEnum $ bitsToInt bits

xor :: Bits -> Bits -> Bits

xor b1 b2 = zipWith xorBit b1 b2

where xorBit = (\b1 b2 -> if b1 /= b2 then B1 else B0)

pseudoRndSymbol' :: Int -> Int -> Int -> (Bits, Int)

pseudoRndSymbol' 0 g \_ = ([], g)

pseudoRndSymbol' n x\_n m = (bit : bits, g)

where bbsR = powMod x\_n 2 m

bit = if even bbsR then B0 else B1

(bits, g) = pseudoRndSymbol' (n-1) bbsR m

pseudoRndSymbol :: Int -> Int -> (Char, Int)

pseudoRndSymbol x\_n m = (bitsToChar bits, g)

where (bits, g) = pseudoRndSymbol' (maxBits - 1) x\_n m

pseudoRndStr :: Int -> Int -> String

pseudoRndStr x\_n m = chr : str

where (chr, g) = pseudoRndSymbol x\_n m

str = pseudoRndStr g m

enAndDecode' :: String -> String -> [Bits]

enAndDecode' pad text = zipWith xor padB textB

where padB = map charToBits pad

textB = map charToBits text

enAndDecode :: String -> String -> String

enAndDecode pad text = map bitsToChar bits

where bits = enAndDecode' pad text

main :: IO ()

main = do

let p = 6025631

let q = 8478859

let n = 10

let k = 3

let secret = 1141

rnd <- rand

let gen = mkStdGen rnd

let keys = keysGen secret n k gen

text <- readFile "test.txt"

let rndStr = pseudoRndStr secret (p \* q)

let enText = enAndDecode rndStr text

-- writeFile "encode.txt" enText

-- putStrLn enText

let secretRec = secretRecovery $ take k keys

let rndStr2 = pseudoRndStr secretRec (p \* q)

let deText = enAndDecode rndStr enText

writeFile "decode.txt" deText

putStrLn $ show $ length enText

# **Задания 5**

Атака "человек посередине" (MitM) — это тип кибератаки, при которой злоумышленник перехватывает связь между двумя сторонами и выдает себя за одну из них. Это позволяет злоумышленнику перехватывать, изменять или блокировать сообщения, передаваемые между двумя сторонами.

Атаки MitM могут быть реализованы различными способами, в том числе с использованием:

• Атаки ARP-спуфинга: Злоумышленник отправляет поддельные пакеты ARP в сеть, чтобы связать свой MAC-адрес с IP-адресом целевой машины.

• Атаки DNS-спуфинга: Злоумышленник перехватывает запросы DNS и направляет их на поддельные серверы DNS, которые возвращают IP-адреса, контролируемые злоумышленником.

• Атаки перехвата сессий SSL/TLS: Злоумышленник использует поддельные сертификаты или другие техники, чтобы перехватить зашифрованные сеансы SSL/TLS.

Атаки MitM могут иметь серьезные последствия, в том числе:

• Перехват данных: Злоумышленник может перехватывать конфиденциальные данные, такие как имена пользователей, пароли и номера кредитных карт.

• Изменение данных: Злоумышленник может изменять данные, передаваемые между двумя сторонами, что может привести к финансовым потерям или другим проблемам.

• Блокировка связи: Злоумышленник может блокировать связь между двумя сторонами, что может нарушить работу бизнеса или личные коммуникации.

Существует несколько мер, которые можно предпринять для защиты от атак MitM, в том числе:

• Использование двухфакторной аутентификации: Это затрудняет для злоумышленника доступ к учетным записям, даже если они перехватили пароль.

• Использование SSL/TLS: Шифрование связи затрудняет для злоумышленника перехват данных.

• Использование брандмауэров и антивирусного программного обеспечения: Это может помочь предотвратить атаки ARP-спуфинга и другие типы атак MitM.

• Регулярное обновление программного обеспечения: Обновления часто включают исправления для уязвимостей, которые могут быть использованы в атаках MitM.

# **Задание 6**

Стандарты современной криптографии в Российской Федерации

В Российской Федерации криптографические стандарты регулируются Федеральной службой по техническому и экспортному контролю (ФСТЭК России). Основные стандарты современной криптографии, действующие в РФ, включают:

• ГОСТ Р 34.10-2012: Стандарт шифрования данных. Определяет алгоритм симметричного блочного шифрования "Магма".

• ГОСТ Р 34.11-2012: Стандарт электронной подписи. Определяет алгоритмы электронной подписи на основе эллиптических кривых "ЭЦП ГОСТ Р 34.10-2012" и "ЭЦП ГОСТ Р 34.10-2012 с использованием ГОСТ Р 34.11-2012".

• ГОСТ Р 34.12-2015: Стандарт генератора псевдослучайных чисел. Определяет алгоритмы генерации псевдослучайных чисел на основе блочного шифрования "Магма".

• ГОСТ Р 34.13-2015: Стандарт хэширования. Определяет алгоритмы хэширования "Стрибог" и "Стрибог-256".

• ГОСТ Р 34.14-2017: Стандарт протоколов обмена ключами. Определяет протоколы обмена ключами на основе эллиптических кривых "ГОСТ Р 34.10-2012 с использованием ГОСТ Р 34.14-2017" и "ГОСТ Р 34.11-2012 с использованием ГОСТ Р 34.14-2017".

Алгоритм ГОСТ 28147-89, применяемый для шифрования информации, является актуальным на сегодняшний день. Также используются более современные стандарты: ГОСТ Р 34.12-2015, посвященный блочным шифрам, а режимы их работы регулируются ГОСТ Р 34.13-2015.

Стандарт ГОСТ 28147-89 регламентирует аспекты симметричного блочного шифрования, а также предписывает возможные режимы его использования. Относится к DES-подобным криптосистемам, при шифровании применяется схема Фейстеля. Разработка этого стандарта шифрования стартовала в 1976 году, а нормативный акт вступил в силу 1 июля 1990 года.