МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

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

Кафедра защиты информации

**

**ОТЧЁТ**

**по практической работе №2**

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

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

Выполнил:Проверил:

Студент гр. «АБс-324», «АВТФ» *Доцент ЗИ*

*Бурлаков И. Е. Архипова А. Б.*

«\_\_» \_\_\_\_\_\_ 2024 г.«\_\_\_» \_\_\_\_\_\_ 2024 г.

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

(подпись) (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.

**Задания:**

**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).

**Задание 2.** Реализовать обобщенный алгоритм Евклида для вычисления с\*d mod m=1.

**Задание 3.** Реализовать расширенный алгоритм Евклида для вычисления взаимообратного числа с -1 mod m=d.

**Задание 4.** Написать программу, использующую алгоритм шифрования данных для преобразования исходного текста. **Вариант 5 - Хьюза (Hughes)**.

**Задание 5.** Найти последнюю цифру «трехэтажного числа».

**Задание 6.** Написать сообщение на тему «Атака посередине». Можно продемонстрировать эмуляцию атаки на базе программы задания 4 данной практической работы.

**Задание 7.** Написать сообщение на тему «Стандарты современной криптографии в РФ».

**Методика выполнения работы**:

1. Разработать алгоритм решения задачи по индивидуальному заданию.
2. Написать и отладить программу решения задачи.
3. Протестировать работу программы на различных исходных данных.
4. Ответить на вопросы по выполненным заданиям, по запросу преподавателя модифицировать код.
5. Ответить на вопросы согласно списку понятий к защите практики (с численными примерами).
6. По запросу преподавателя решить практическое задание на тему «Изучение математических основ криптографии».

**Задания**

# 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. через разложение степени в двоичный вид (логарифм).

## C++:

// Теорема Ферма и св-во сравнений(2 алгоритма)

#include <iostream>

#include <math.h>

using namespace std;

bool CheckPrime(int p) { // проверка, является ли модуль простым числом

if (p <= 1) return false;

for (int i = 2; i \* i <= p; ++i) {

if (p % i == 0) return false;

}

return true;

}

int algorithm1(int a, int x, int p) { // Алгоритм1 через уменьшение степени

int result = 1;

a %= p;

while (x > 0) {

if (x % 2 == 1)

result = (result \* a) % p;

x /= 2;

a = (a \* a) % p;

}

return result;

}

int algorithm2(int a, int x, int p) { // Алгоритм2 через логарифм

int nums[100]; // инициализируем массив чисел, в котором хранится числовой ряд a

int stepen = 1; // для проверки, чтобы степень не превышала себя

nums[0] = a % p;

int n=1;

while (stepen <= x) { // вычисляем числовой ряд для числа a

stepen \*= 2;

nums[n] = (nums[n-1] \* nums[n-1]) % p;

n += 1;

}

int binary[100]; // инициализируем массив чисел, в котором хранится число в двоичной степени

n = 0;

while (x != 1) { // переводчим степень в двоичное число

binary[n] = x % 2;

x /= 2;

n += 1;

}

binary[n] = x;

n += 1;

int result = 1;

for (int i = 0; i < n; ++i) { // используем конечную формулу алгоритма

result \*= pow(nums[i], binary[i]);

}

result %= p;

return result;

}

int main(){

setlocale(LC\_ALL, "Rus");

cout << "Нахождение остатка числа a^x mod p:" << endl;

int a, x, p;

cout << "Введите числа a, x и p: ";

cin >> a >> x >> p;

if (CheckPrime(p)==false) { // проверка, является ли модуль простым числом

cout << "p не простое число." << endl;

return 1;

}

if (a % p == 0 && x != 0) { // проверка, выполняется ли теорема Ферма

cout << "Теорема Ферма не выполняется: a^x ≡ 0 (mod p)" << endl;

return 1;

}

int result1 = algorithm1(a, x, p);

cout << "Алгоритм 1: " << a << "^" << x << " mod " << p << " = " << result1 << endl;

int result2 = algorithm2(a, x, p);

cout << "Алгоритм 2: " << a << "^" << x << " mod " << p << " = " << result2 << endl;

return 0;

}

Input: Сравнение по модулю простого числа a^x mod p:

Введите числа a, x и p: 7 13 5

7^13 mod 5 = 2

## Python:

# Теорема Ферма и св-во сравнений(2 алгоритма)

def CheckPrime(p): # проверка, является ли модуль простым числом

if (p <= 1): return False

for i in range(2, int(p \*\* 0.5) + 1):

if (p % i == 0): return False

return True

def algorithm1(a, x, p): # Алгоритм1 через уменьшение степени

result = 1

a %= p

while (x > 0):

if (x % 2 == 1):

result = (result \* a) % p

x //= 2

a = (a \* a) % p

return result

def algorithm2(a, x, p):

nums = [0] \* 100 # инициализируем массив чисел, в котором хранится числовой ряд a

stepen = 1 # для проверки, чтобы степень не превышала себя

nums[0] = a % p

n = 1

while (stepen <= x): # вычисляем числовой ряд для числа a

stepen \*= 2

nums[n] = (nums[n-1] \* nums[n-1]) % p

n += 1

binary = [0] \* 100 # инициализируем массив чисел, в котором хранится число в двоичной степени

n = 0

while (x != 1): # переводчим степень в двоичное число

binary[n] = x % 2

x //= 2

n += 1

binary[n] = x

n += 1

result = 1

for i in range(n): # используем конечную формулу алгоритма

result \*= pow(nums[i], binary[i])

result %= p

return result

print("Нахождение остатка числа a^x mod p:")

a = int(input("Введите число а: "))

x = int(input("Введите степень x: "))

p = int(input("Введите модуль p: "))

if (CheckPrime(p) == False): print("p не простое число.") # проверка, является ли модуль простым числом

elif (a % p == 0) and (x != 0): print("Теорема Ферма не выполняется: a^x ≡ 0 (mod p)") # проверка, выполняется ли теорема Ферма

else:

result1 = algorithm1(a, x, p)

print(f"Алгоритм 1: {a}^{x} mod {p} = {result1}")

result2 = algorithm2(a, x, p)

print(f"Алгоритм 2: {a}^{x} mod {p} = {result2}")

Input: Сравнение по модулю простого числа a^x mod p:

Введите число а: 7

Введите степень x: 13

Введите модуль p: 5

7^13 mod 5 = 2

# Задание 2. Реализовать обобщенный алгоритм Евклида для вычисления с\*d mod m=1.

## С++:

#include <iostream>

#include <vector>

using namespace std;

void algorithm(int c, int m, int &gcd, int &d) { // Алгоритм нахождения НОД(c, m) и числа d

int q;

vector<int> one = {m, 0};

vector<int> two = {c, 1};

vector<int> three(2);

while (two[0] != 0) {

q = one[0] / two[0];

three[0] = one[0] % two[0];

three[1] = one[1] - (q \* two[1]);

one[0] = two[0]; one[1] = two[1];

two[0] = three[0]; two[1] = three[1];

}

gcd = one[0], d = one[1];

}

int main(){

setlocale(LC\_ALL, "Rus");

cout << "Нахождение d, чтобы c\*d mod m = 1:" << endl;

int c, m;

cout << "Введите числа c и m: ";

cin >> c >> m;

int gcd, d;

algorithm(c, m, gcd, d);

if (gcd != 1) {

cout << "Такого d не существует, т.к. c и m не взаимнопростые.";

} else {

if (d < 0) {

d += m;

}

cout << "Число d: " << d << endl << c\*d << " mod " << m << " = " << (c\*d) % m;

}

return 0;

}

Input: Нахождение d, чтобы c\*d mod m = 1:

Введите числа c и m: 7 5

Число d: 3

21 mod 5 = 1

## Python:

def algorithm(c, m): # Алгоритм нахождения НОД(c, m) и числа d

one = [m, 0]

two = [c, 1]

three = [0, 0]

q = 0

while (two[0] != 0):

q = one[0] // two[0]

three[0] = one[0] % two[0]

three[1] = one[1] - (q \* two[1])

one[0], one[1] = two[0], two[1]

two[0], two[1] = three[0], three[1]

return one[0], one[1]

print("Нахождение d, чтобы c\*d mod m = 1:")

c = int(input("Введите число c: "))

m = int(input("Введите число m: "))

gcd, d = algorithm(c, m)

if (gcd != 1):

print("Такого d не существует, т.к. c и m не взаимнопростые.")

else:

if (d < 0):

d += m

print("Число d:", d)

print(c\*d, "mod", m, "=", (c\*d) % m)

Input: Нахождение d, чтобы c\*d mod m = 1:

Введите число c: 7

Введите число m: 5

Число d: 3

21 mod 5 = 1

# Задание 3. Реализовать расширенный алгоритм Евклида для вычисления взаимообратного числа с -1 mod m=d.

## C++:

#include <iostream>

#include <vector>

using namespace std;

int gcd(int c, int m) { // нахождение НОД(c, m)

if (m == 0) return c;

return gcd(m, c % m);

}

int main(){

setlocale(LC\_ALL, "Rus");

cout << "Реализация расширенного алгоритм Евклида для вычисления взаимообратного числа с^-1 mod m = d." << endl;

int c, m;

cout << "Введите числа c и m: ";

cin >> c >> m;

if (gcd(c, m) != 1) {

cout << "Числа не взаимообратные.";

return 1;

}

// реализация расширенного алгоритма и вывод его значений

cout << "Табличное представление данных расширенного алгоритма Евклида: " << endl;

cout << "A | " << c << " " << "1 " << "0" << endl;

cout << "B | " << m << " " << "0 " << "1" << endl;

int q;

vector<int> one = {c, 1, 0};

vector<int> two = {m, 0, 1};

vector<int> three(3);

while (two[0] != 0) {

q = one[0] / two[0];

three[0] = one[0] % two[0];

three[1] = one[1] - (q \* two[1]);

three[2] = one[2] - (q \* two[2]);

one = two;

two = three;

cout << "T | " << three[0] << " " << three[1] << " " << three[2] << " " << q << endl;

}

if (one[1] < 0) {

one[1] += m;

}

cout << "Взаимно обратное: " << one[1];

return 0;

}

Input: Реализация расширенного алгоритм Евклида для вычисления взаимообратного числа с^-1 mod m = d.

Введите числа c и m: 7 5

Табличное представление данных расширенного алгоритма Евклида:

A | 7 1 0

B | 5 0 1

T | 2 1 -1 1

T | 1 -2 3 2

T | 0 5 -7 2

Взаимно обратное: 3

## Python:

def gcd(c, m): # нахождение НОД(c, m)

if (m == 0): return c

return gcd(m, c % m)

print("Реализовать расширенный алгоритм Евклида для вычисления взаимообратного числа s^-1 mod m = d.")

c = int(input("Введите число c: "))

m = int(input("Введите число m: "))

if (gcd(c, m) != 1):

print("Числа не взаимообратные.")

exit(1)

# реализация расширенного алгоритма и вывод его значений

print("Табличное представление данных расширенного алгоритма Евклида:")

print(f"A | {c} 1 0")

print(f"B | {m} 0 1")

q = 0

one = [c, 1, 0]

two = [m, 0, 1]

three = [0, 0, 0]

while (two[0] != 0):

q = one[0] // two[0]

three[0] = one[0] % two[0]

three[1] = one[1] - (q \* two[1])

three[2] = one[2] - (q \* two[2])

one = two.copy()

two = three.copy()

print(f"T | {three[0]} {three[1]} {three[2]} {q}")

if (one[1] < 0):

one[1] += m

print(f"Взаимно обратное: {one[1]}")

Input: Реализовать расширенный алгоритм Евклида для вычисления взаимообратного числа s^-1 mod m = d.

Введите число c: 7

Введите число m: 5

Табличное представление данных расширенного алгоритма Евклида:

A | 7 1 0

B | 5 0 1

T | 2 1 -1 1

T | 1 -2 3 2

T | 0 5 -7 2

Взаимно обратное: 3

# Задание 4. Написать программу, использующую алгоритм шифрования данных для преобразования исходного текста. Вариант 5 - Хьюза (Hughes).

## C++:

#include <iostream>

#include <time.h>

#include <cmath>

#include <vector>

using namespace std;

// нахождение НОД(a, b) для проверки, являются ли числа взаимопростыми

int VzaimProst(int a, int b) {

if (b == 0) return a;

return VzaimProst(b, a % b);

}

// проверка, является ли число простым

bool CheckPrime(int p) {

if (p <= 1) return false;

for (int i = 2; i \* i <= p; ++i)

{

if (p % i == 0) return false;

}

return true;

}

// функция Эйлера

int fi(int num) {

int result = 1;

int stepen = 0;

for (int i = 2; i <= num; ++i)

{

while (num % i == 0)

{

stepen++;

num /= i;

}

if (stepen != 0) {

result \*= pow(i, stepen - 1) \* (i - 1);

}

stepen = 0;

}

return result;

}

// нахождение остатка от числа в степени

int Pow(int a, int x, int p)

{

int result = 1;

if (CheckPrime(p) == true)

{ // простое

x %= p - 1;

}

else { // не простое

x %= fi(p);

}

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

result = (result \* a) % p;

}

return result;

}

// Алгоритм нахождения числа d(c^-1 mod m)

int inversia(int c, int m)

{

int q;

vector<int> one = { m, 0 };

vector<int> two = { c, 1 };

vector<int> three(2);

while (two[0] != 0)

{

q = one[0] / two[0];

three[0] = one[0] % two[0];

three[1] = one[1] - (q \* two[1]);

one[0] = two[0]; one[1] = two[1];

two[0] = three[0]; two[1] = three[1];

}

if (one[1] < 0)

{

one[1] += m;

}

return one[1];

}

int main() {

setlocale(LC\_ALL, "Rus");

srand(time(0));

string message;

cout << "Введите сообщение: ";

cin >> message;

int g, n;

cout << "Введите числа g и n: ";

cin >> g >> n;

int x = 100 + rand() % 100000; // абонент A выбирает случайное больше число и генерирует закрытый ключ

cout << "x = " << x << endl;

int k = Pow(g, x, n);

cout << "k = " << k << endl;

int y = 100 + rand() % 100000; // абонент B выбирает случайное больше число

while (VzaimProst(y, n - 1) != 1)

{

y = 100 + rand() % 100000;

}

cout << "y = " << y << endl;

int Y = Pow(g, y, n); // абонент B посылает абоненту A

cout << "Y = " << Y << endl;

int X = Pow(Y, x, n); // абонент А посылает абоненту B

cout << "X = " << X << endl;

int z = inversia(y, n - 1); // абонент B вычисляет

cout << "z = " << z << endl;

int ksh = Pow(X, z, n);

cout << "k' = " << ksh << endl;

if (k == ksh)

{

int key = k;

for (int i = 0; i < message.length(); ++i)

{

message[i] = message[i] ^ key;

}

cout << "Зашифрованное сообщение: " << message << endl;

for (int i = 0; i < message.length(); ++i)

{

message[i] = message[i] ^ key;

}

cout << "Расшифрованное сообщение: " << message;

}

else

{

cout << "Ключи не равны!";

}

return 0;

}

Input: Введите сообщение: Novosibirsk

Введите числа g и n: 67 13

x = 4166

k = 4

y = 11419

Y = 11

X = 4

z = 7

k' = 4

Зашифрованное сообщение: Jkrkwmfmvwo

Расшифрованное сообщение: Novosibirsk

## Python:

import random

def VzaimProst(a, b): # нахождение НОД(a, b) для проверки, являются ли числа взаимопростыми

if (b == 0): return a

return VzaimProst(b, a % b)

def CheckPrime(p): # проверка, является ли число простым

if (p <= 1): return False

for i in range(2, int(p \*\* 0.5) + 1):

if (p % i == 0): return False

return True

def fi(num): # функция Эйлера

result = 1

stepen = 0

for i in range(2, num+1):

while (num % i == 0):

stepen += 1

num //= i

if (stepen != 0):

result \*= pow(i, stepen - 1) \* (i - 1)

stepen = 0

return result

def Pow(a, x, p): # нахождение остатка от числа в степени

result = 1

if (CheckPrime(p) == True): x %= p - 1 # простое

else: x %= fi(p) # не простое

for i in range(1, x+1):

result = (result \* a) % p

return result

def inversia(c, m): # алгоритм нахождения числа d(c^-1 mod m)

q = 0

one = [m, 0]

two = [c, 1]

three = [0, 0]

while (two[0] != 0):

q = one[0] // two[0]

three[0] = one[0] % two[0]

three[1] = one[1] - (q \* two[1])

one[0], one[1] = two[0], two[1]

two[0], two[1] = three[0], three[1]

if (one[1] < 0):

one[1] += m

return one[1]

message = input("Введите сообщение: ")

g = int(input("Введите число g: "))

n = int(input("Введите число n: "))

x = random.randrange(10000, 100000) # абонент A выбирает случайное больше число и генерирует закрытый ключ

print("x =", x)

k = Pow(g, x, n)

print("k =", k)

y = random.randrange(10000, 100000) # абонент B выбирает случайное больше число

while VzaimProst(y, n-1) != 1:

y = random.randrange(10000, 100000)

print("y =", y)

Y = Pow(g, y, n) # абонент B посылает абоненту A

print("Y =", Y)

X = Pow(Y, x, n) # абонент А посылает абоненту B

print("X =", X)

z = inversia(y, n-1) # абонент B вычисляет

print("z =", z)

ksh = Pow(X, z, n)

print("k' =", ksh)

if (k == ksh):

key = k

ArrEncrypted = []

ArrDecrypted = []

Encrypted = ""

Decrypted = ""

for i in range(len(message)): # Шифрование

ArrEncrypted.append(chr(ord(message[i]) ^ key))

for i in range(len(ArrEncrypted)):

Encrypted += ArrEncrypted[i]

print("Зашифрованное сообщение:", Encrypted)

for i in range(len(Encrypted)): # Дешифрование

ArrDecrypted.append(chr(ord(Encrypted[i]) ^ key))

for i in range(len(ArrDecrypted)):

Decrypted += ArrDecrypted[i]

print("Расшифрованное сообщение:", Decrypted)

else: print("Ключи не равны!")

Input: Введите сообщение: Novosibirsk

Введите число g: 67

Введите число n: 13

x = 63462

k = 12

y = 44119

Y = 11

X = 12

z = 7

k' = 12

Зашифрованное сообщение: Bczcene~g

Расшифрованное сообщение: Novosibirsk

# Задание 5. Найти последнюю цифру «трехэтажного числа».

## C++:

#include <iostream>

#include <math.h>

using namespace std;

int LastNum(int num, int nums[]) { // нахождение уникальной комбинации последних цифр в числе

int k = 0, n = 0; // k - условие выхода из цикла / n - счётчик

for (int i = 0; k == 0; i++) {

nums[i] = pow(num, i+1);

nums[i] = nums[i] % 10;

if (nums[i] == nums[i-1] || nums[i] == nums[i-2] || nums[i] == nums[i-3] || nums[i] == nums[i-4]) {

k = k + 1;

} else {

n = n + 1;

}

}

return n;

}

int main(){

setlocale(LC\_ALL, "Rus");

int a, b, c;

cout << "Введите число, первую и вторую степень: ";

cin >> a >> b >> c;

a = a % 10;

int nums1[10];

int n1;

n1 = LastNum(a, nums1); // находим уникальные конечные цифры и их кол-во

int step1 = b % n1;

if (step1 == 0) {

b = nums1[n1-1]; // если остаток = 0, то конечная цифра - последняя цифра комбинации

} else {

b = nums1[step1-1]; // иначе (цифра комбинации) - 1

}

int nums2[10];

int n2;

n2 = LastNum(b, nums2);

int step2 = c % n2;

if (step2 == 0) {

c = nums2[n2-1];

} else {

c = nums2[step2-1];

}

cout << "Последняя цифра: " << c; // вывод последней цифры числа

return 0;

}

Input:

Введите число, первую и вторую степень: 3 7 8

Последняя цифра числа: 1

## Python:

def LastNum(num, nums): # нахождение уникальной комбинации последних цифр в числе

k = 0 # условие выхода из цикла

n = 0 # счётчик

i = 0

while (k == 0):

nums[i] = pow(num, i+1)

nums[i] = nums[i] % 10

if nums[i] == nums[i-1] or nums[i] == nums[i-2] or nums[i] == nums[i-3] or nums[i] == nums[i-4]: k += 1

else: n += 1

i += 1

return n

a = int(input("Введите число: "))

b = int(input("Введите 1-ю степень: "))

c = int(input("Введите 2-ю степень: "))

a = a % 10

nums1 = [0] \* 10

n1 = LastNum(a, nums1) # находим уникальные конечные цифры и их кол-во

step1 = b % n1

if (step1 == 0): b = nums1[n1-1] # если остаток = 0, то конечная цифра - последняя цифра комбинации

else: b = nums1[step1-1] # иначе (цифра комбинации) - 1

nums2 = [0] \* 10

n2 = LastNum(b, nums2)

step2 = c % n2

if (step2 == 0): c = nums2[n2-1]

else: c = nums2[step2-1]

print("Последняя цифра:", c)

Input:

Введите число: 3

Введите 1-ю степень числа: 7

Введите 2-ю степень числа: 8

Последняя цифра: 1

# Задание 6. Написать сообщение на тему «Атака посередине».

**Атака "Встреча посередине"** - это тип криптоаналитической атаки, обычно используемой против алгоритмов шифрования, использующих симметричные ключи. Кроме того, атака использует тот факт, что процессы шифрования и дешифрования являются обратными операциями друг друга. Таким образом, злоумышленник использует метод грубой силы, чтобы найти секретный ключ, который мы используем в процессе шифрования.

Атака работает путем разделения пространства ключей на две части. Во-первых, она шифрует открытый текст, используя все возможные ключи из половины пространства ключей. Кроме того, злоумышленник расшифровывает полученный зашифрованный текст, используя все возможные ключи из другой половины пространства ключей. Наконец, злоумышленник проверяет соответствие промежуточных значений, полученных на этапах шифрования и дешифрования.

Если найдено совпадение, ключ, используемый при шифровании и дешифровании, находится в середине ключевого пространства. Повторяя процесс с меньшими пробелами по обе стороны от соответствующего ключа, злоумышленник может в конечном итоге найти секретный ключ, используемый при шифровании. Атака "Встреча посередине" особенно эффективна против алгоритмов шифрования с относительно небольшим пространством ключей и уязвима для исчерпывающего поиска ключей.

**Порядок работы:**

Злоумышленник обычно выполняет пять шагов, чтобы реализовать или запустить атаку "встреча посередине":

Первым шагом является разделение пространства ключей. Злоумышленник делит пространство ключей на две части, каждая из которых содержит половину возможных ключей. Кроме того, разделение пространства ключей имеет решающее значение для успеха атаки. Вероятность успешной реализации MITM-атаки высока, если пространство ключей невелико.

Затем злоумышленник шифрует открытый текст, используя все возможные ключи из половины пространства ключей. Кроме того, он сохраняет полученные промежуточные значения в таблице.

Продвигаясь вперед, злоумышленник расшифровывает зашифрованный текст, используя все возможные ключи из другой половины пространства ключей, и сохраняет полученные промежуточные значения в отдельной таблице.

Следующим шагом является сопоставление промежуточных значений из двух таблиц. Злоумышленник сравнивает промежуточные значения на этапах шифрования и дешифрования, чтобы увидеть, есть ли совпадение. Если мы находим совпадение, соответствующий ключ находится в середине ключевого пространства.

Наконец, злоумышленник повторяет процесс с меньшими пробелами по обе стороны от соответствующего ключа, пока не будет найден используемый секретный ключ.

Важно отметить, что атака "встреча посередине" требует значительных вычислительных мощностей и памяти. Следовательно, это может быть невозможно для алгоритмов шифрования с большими пространствами ключей или другими строгими мерами безопасности.

MITM-атаки обычно применимы только к блочным шифрам и могут быть неэффективны против потоковых шифров. Таким образом, MITM-атаки могут быть эффективными в определенных ситуациях, но они не являются надежным методом взлома шифрования.

**Влияние:**

Влияние атаки "встреча посередине" может быть значительным. Давайте обсудим некоторые последствия.

Злоумышленник может получить секретный ключ для шифрования, а также расшифровать сообщения, запустив MITM-атаку. Это может позволить злоумышленнику расшифровывать другие сообщения, зашифрованные тем же ключом.

Более того, такая атака может привести к утечке конфиденциальной информации, что может иметь ряд негативных последствий, в зависимости от характера информации и от того, кто имеет к ней доступ.

Используя MITM-атаку, злоумышленник может скомпрометировать систему, используемую для шифрования и дешифрования сообщений. Это может позволить злоумышленнику получить доступ к сети, а также к системам, подключенным к сети. Более того, злоумышленник может установить бэкдоры для выполнения вредоносных действий.

Наконец, MITM-атака может нанести ущерб репутации системы или организации. Это может привести к потере доверия пользователей, заказчиков и партнеров.

**Методы предотвращения:**

Мы можем использовать несколько механизмов предотвращения, чтобы снизить риск атаки "встреча посередине" (MITM). Давайте обсудим некоторые основные методы.

Одним из ключевых недостатков MITM-атаки является то, что она требует больших вычислительных мощностей для поиска по большому количеству возможных ключей. При использовании более длинных ключей количество возможных ключей увеличивается экспоненциально, что затрудняет поиск методом перебора.

Кроме того, некоторые шифры более уязвимы для MITM-атак. Используя шифры, устойчивые к MITM-атакам, мы можем эффективно снизить риск атаки.

При атаке MITM злоумышленник ищет ключ, который расшифровывает зашифрованный текст в открытый. Следовательно, мы можем использовать случайные ключи, чтобы уменьшить вероятность того, что злоумышленник угадает правильный ключ.

Более того, протоколы обмена ключами, такие как Диффи-Хеллман, позволяют двум сторонам безопасно согласовать общий секретный ключ без его передачи. Это может затруднить злоумышленнику перехват ключа и запуск MITM-атаки.

Наконец, мы можем реализовать аутентификацию сообщения. Аутентификация сообщения гарантирует, что сообщение не было подделано во время передачи. Это может затруднить злоумышленнику изменение сообщения и запуск MITM-атаки.

# Задание 7. Написать сообщение на тему «Стандарты современной криптографии в РФ».

Начавшееся в конце ХХ в. активное внедрение информационно-телекоммуникационных технологий в деятельность государственных и коммерческих организаций, в свою очередь, потребовало использования средств криптографической защиты информации для обеспечения безопасности хранимых и передаваемых данных. В основе средств криптографической защиты информации лежат базовые криптографические алгоритмы, с использованием которых строятся реализуемые средством протоколы и сервисы. Утвержденное Приказом ФСБ России от 9 февраля 2005 г. № 66 "Положение о разработке, производстве, реализации и эксплуатации шифровальных (криптографических) средств защиты информации" (Положение ПКЗ-2005) рекомендует использовать при разработке средств криптографической защиты информации криптографические алгоритмы, утвержденные в качестве национальных стандартов. Однако, в соответствии с мировой практикой, сами стандарты в области криптографии содержат описание алгоритмов функционирования криптографических механизмов, но не описывают прикладные протоколы информационного взаимодействия, что, в частности, может вызвать серьезные проблемы при обеспечении корректной встречной работы средств криптографических защиты информации различных производителей.

С целью обеспечения деятельности по разработке криптографических стандартов и нормативных документов, регламентирующих их применение, Приказом Ростехрегулирования от 28 декабря 2007 г. был создан технический комитет по стандартизации "Криптографическая защита информации", получивший сокращенное наименование ТК 26. Этим приказом были утверждены положение о техническом комитете, его структура и перечень организаций (предприятий), ставших его членами.

На момент создания ТК 26 в Российской Федерации действовало четыре национальных стандарта в области криптографической защиты информации:

ГОСТ 28147–89, определяющий алгоритм криптографического преобразования (блочный шифр) и режимы работы данного алгоритма;

ГОСТ Р 34.11–94, определяющий криптографическую функцию хеширования;

ГОСТ Р 34.10–2001, определяющий алгоритмы формирования и проверки электронной цифровой подписи;

ГОСТ Р ИСО/МЭК 10116–93, определяющий режимы работы блочного алгоритма шифрования.

Количество организаций, заинтересованных в деятельности ТК 26, ежегодно растет, и за годы его существования число членов технического комитета увеличилось с 44 до 61. Среди них – федеральные органы исполнительной власти, научные учреждения, предприятия – разработчики СКЗИ, испытательные лаборатории, системные интеграторы. Таким образом, ТК 26 является постоянно действующим коллегиальным органом, организующим разработку и экспертизу проектов национальных, межгосударственных и международных стандартов в закрепленной области деятельности, обеспечивающим анализ технических регламентов и стандартов на предмет их обновления и дальнейшего использования.

Активное развитие вычислительной техники, методов синтеза и анализа криптографических алгоритмов, а также задач, возникающих перед предприятиями-разработчиками при создании новых типов СКЗИ, потребовало обновления системы отечественных криптографических стандартов. В связи с этим в 2010–2015 гг. ТК 26 проведен комплекс мероприятий по обновлению национальных стандартов в области криптографии:

приказом Федерального агентства по техническому регулированию и метрологии от 7 августа 2012 г. № 215-ст утвержден ГОСТ Р 34.10–2012 "Информационная технология. Криптографическая защита информации. Процессы формирования и проверки электронной цифровой подписи";

приказом Федерального агентства по техническому регулированию и метрологии от 7 августа 2012 г. № 216-ст утвержден ГОСТ Р 34.11–2012 "Информационная технология. Криптографическая защита информации. Функция хеширования".

в настоящее время разработаны, одобрены ТК 26 и готовятся к утверждению стандарты, определяющие блочные шифры и режимы их работы. В первом из них существующий стандарт (ГОСТ 28147–89) будет дополнен новым алгоритмом с длиной блока 128 бит, а для стандартизированного в настоящее время алгоритма с длиной блока 64 бита будет зафиксирован набор подстановок. Второй стандарт будет содержать описание набора основных режимов работы блочных шифров, необходимых для разработки СКЗИ.

Однако, как мы уже отметили ранее, криптографический стандарт интересен для разработчика средств защиты только в том случае, если имеется возможность его реализации в используемых криптографических протоколах информационного взаимодействия. В связи с этим в настоящее время значительные усилия ТК 26, инициированные прежде всего организациями – разработчиками СКЗИ, направлены на разработку методических документов, касающихся вопросов непосредственного применения стандартизированных криптографических алгоритмов.

В частности, на сегодняшний день утверждены методические рекомендации и технические спецификации, регламентирующие использование национальных криптографических стандартов в профиле сертификата и списке отзыва сертификатов (CRL) инфраструктуры открытых ключей X.509, криптографических сообщениях формата CMS; протоколе безопасности транспортного уровня (TLS), шифровании вложений в протоколе IPSEC ESP. Отдельными документами регламентируются вопросы задания узлов замены блока подстановок алгоритма шифрования ГОСТ 28147–89 и параметров эллиптических кривых в соответствии с ГОСТ Р 34.10–2012. Описаны криптографические алгоритмы, сопутствующие применению стандартов ГОСТ Р 34.10–2012 и ГОСТ Р 34.11–2012 в современных протоколах защищенного информационного взаимодействия, а также дополнения к стандартам PKCS#5, PKCS#8 и PKCS#15, регламентирующие использование отечественных криптографических алгоритмов.

В настоящее время усиленно ведутся работы по описанию расширения PKCS#11 для использования российских стандартов ГОСТ Р 34.10–2012 и ГОСТ Р 34.11–2012, а также по интеграции отечественных криптографических алгоритмов в протоколы IPSec, IKE, IPlir, SCP-F2.

На всех этапах работ по разработке национальных стандартов и методических документов проводится всестороннее исследование предлагаемых к стандартизации криптографических решений на основе научно-технической экспертизы и публичных обсуждений. Такой подход позволяет, с одной стороны, провести обоснованную оценку криптографических характеристик предлагаемых решений, а с другой – учесть пожелания организаций-разработчиков в части, касающейся эксплуатационных свойств.

Исследования, проводимые по данным направлениям, показывают заинтересованность отечественных компаний в развитии защищенных информационных технологий, основанных на российских криптографических стандартах. Деятельность, проводимая членами ТК 26 в данном направлении, позволит в ближайшем будущем обеспечить возможность замены ряда криптографических сервисов, предоставляемых в настоящее время в информационно-телекоммуникационном пространстве с использованием иностранных криптографических алгоритмов, на криптографические сервисы, базирующиеся на российских криптографических алгоритмах.

Деятельность технического комитета активно способствует развитию криптографических исследований: проводимый совместно Математическим институтом им. В.А. Стеклова РАН и Академией криптографии Российской Федерации ежегодный симпозиум "Современные тенденции в криптографии" (CTCrypt) с каждым годом привлекает все больше специалистов из российских и зарубежных научных, образовательных и коммерческих организаций.

Открытый конкурс научно-исследовательских работ, посвященных анализу криптографических качеств хеш-функции ГОСТ Р 34.11–2012, который проводился в 2014–2015 гг., позволил привлечь внимание международной и российской научной общественности к российским криптографическим алгоритмам и получить независимые оценки их криптографических характеристик. Лауреатами конкурса являются представители Канады, Китая, Российской Федерации и Сингапура.

Важной составляющей деятельности технического комитета является участие в работе профильной рабочей группы подкомитета "Информационная безопасность" первого объединенного технического комитета Международной организации по стандартизации. В частности, нашими экспертами был выявлен ряд слабостей в международных криптографических стандартах, разработаны предложения, позволившие существенно уточнить положения базовых документов данной рабочей группы в части оценки характеристик криптографических алгоритмов.

С учетом анализа деятельности технического комитета по международной и национальной стандартизации в 2014 г. был разработан и утвержден председателем ТК 26 "Порядок оформления документов, содержащих проекты методических рекомендаций по вопросам, относящимся к стандартизации в области криптографической защиты информации". Данный документ, определяющий набор базовых требований к содержанию, обоснованию и форме первоначальных предложений, поступающих в адрес комитета, призван оптимизировать процедуру разработки и максимально упростить процесс экспертной оценки синтезных решений, рассматриваемых техническим комитетом.

В заключение необходимо отметить, что разработка новых принципов реализации стойких криптографических алгоритмов и конструкций отечественными специалистами, а также поиск новых и перспективных областей применения отечественных криптографических алгоритмов являются приоритетными задачами ТК 26 и его членов. Двери технического комитета открыты для всех, кто заинтересован в развитии системы национальной стандартизации в области криптографической защиты информации.

**Вывод:**

В ходе выполнения практической работы №2 я познакомился с различными алгоритмами, которые пригодятся в дальнейшем изучении криптографии. Научился пользоваться алгоритмом распределения ключей Хьюга. Познакомился с базовой информацией по теме “Криптография”.

**Ссылка на репозиторий:**

https://github.com/Evlusha/PR2.git