МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»

Кафедра теоретических основ компьютерной безопасности и криптографии

ЛАБОРАТОРНАЯ РАБОТА №1
Протокол Диффи - Хеллмана

студента 5 курса 531 группы
специальности 10.05.01 «Компьютерная безопасность»
факультета компьютерных наук и информационных технологий
Никитина Арсения Владимировича

Саратов 2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В данной лабораторной работе поставлена задача рассмотреть базовый протокол Диффи – Хеллмана, на основе полученного материала реализовать схему открытого распределения ключей.

1 Цель работы и порядок ее выполнения

Цель работы – изучение протокола Диффи-Хеллмана и его реализация.

Порядок выполнения работы:

1. Разобрать, что такое протоколы открытого распределения ключей;
2. Разобрать протокол Диффи-Хеллмана;
3. Произвести программную реализацию.

2 Теоретическая часть

Протокол Диффи-Хеллмана – протокол, позволяющий двум и более сторонам получить общий секретный ключ, используя незащищенный от прослушивания канал связи. Полученный ключ используется для шифрования дальнейшего обмена с помощью алгоритмов симметричного шифрования.

Схема открытого распределения ключей, предложенная Диффи и Хеллманом, произвела настоящую революцию в мире шифрования, так как снимала основную проблему классической криптографии – проблему распределения ключей.

Безопасность алгоритма опирается на трудность вычисления дискретных логарифмов в конечном поле (по сравнению с легкостью возведения в степень в том же самом поле, алгоритм может быть использован для распределения ключей – Алиса и Боб могут воспользоваться этим алгоритмом для генерации секретного ключа, но его нельзя использовать для шифрования и дешифрования сообщений.

Алиса и Боб перед непосредственно протоколом выбирают простые числа p и g, такие, что g – первообразный корень p. Эти два целых числа можно не хранить в секрете и, более того, их можно использовать, передавая по не секретному каналу. Эти числа могут даже использоваться группой пользователей.

Непосредственно алгоритм состоит из 4 шагов и выглядит так:

1. Алиса выбирает случайное число a и посылает бобу значение $A = g^a \bmod p$;

2. Боб выбирает случайное число b и посылает Алисе значение $B = g^b \bmod p$;

3. Алиса вычисляет ключ $K_1 = B^a \bmod p$;

4. Боб вычисляет ключ $K_2 = A^b \bmod p$;

$K_1 = K_2 = g^{ab} \bmod p$.

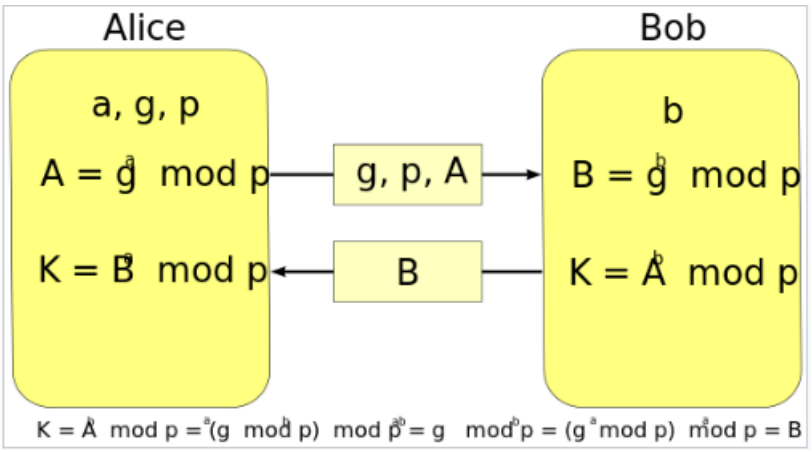

Рисунок 1 – схема протокола

Подслушивающий канал пользователь не может вычислить это значение из известных p, g, A, B: a и b могут быть раскрыты только дискретным логарифмированием (сложная задача), поэтому получаемый ключ является секретным, которые Алиса и Боб вычисляют независимо.

3 Практическая часть

На рисунках 2 – 4 приведены примеры работы программы.

```
PS D:\lab> python crypto1.py
Введите количество бит простого p: 100
Открытый параметр p: 855937819047200022806473750823
Открытый параметр g: 561335737158531531883920397933
Закрытый ключ a: 729317708683087012351671624 68
Закрытый ключ b: 32902749346699543243078428 7017
Открытый ключ Алисы, который передается Бобу:  621153301345117186454085569358
Открытый ключ Боба, который передается Алисе:  297809804280984299022822432997
Вычисляется общий секретный ключ K на стороне Алисы:  654587016292941068217437595905
Вычисляется общий секретный ключ K на стороне Боба:  654587016292941068217437595905
True
```

Рисунок 2 – пример работы программы при длине 100 бит числа p

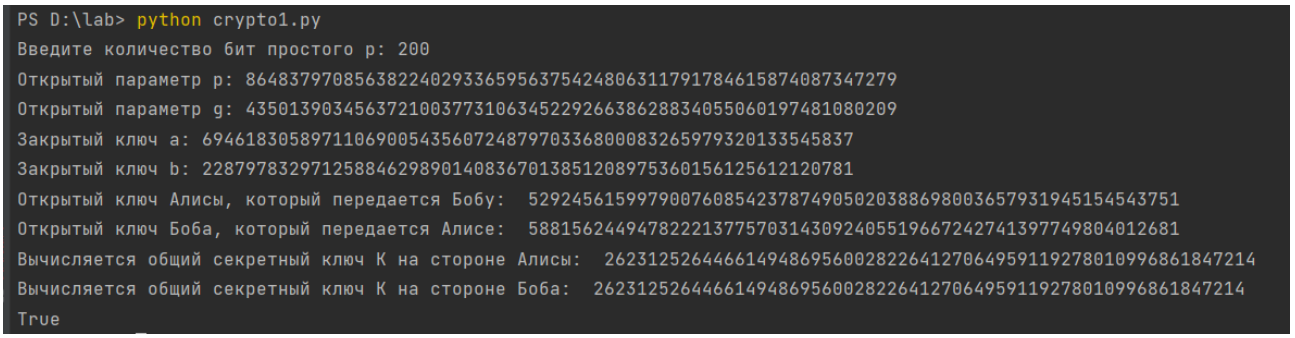

Рисунок 3 – пример работы программы при длине 200 бит числа *p*

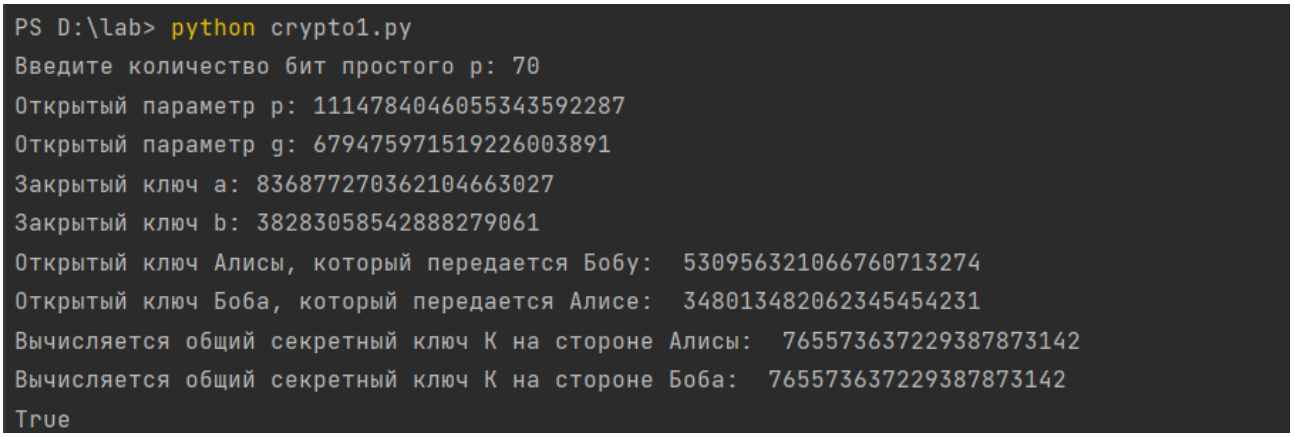

Рисунок 4 – пример работы программы при длине 70 бит числа *p*

ПРИЛОЖЕНИЕ А

Листинг программы

```
import random


def representation(p):

    twos = 0

    while not p % 2:
        twos += 1
        p //= 2

    return twos, p


def miller_rabin_test(n, k):

    s, t = representation(n - 1)

    for _ in range(k):

        a = random.randint(2, n - 2)

        x = pow(a, t, n)

        if x == 1 or x == n - 1:
            continue

        flag = False
        for _ in range(s - 1):

            x = x * x % n

            if x == 1:
                return False

            if x == n - 1:
                flag = True
                break

        if flag:
            continue

        return False

    return True


def get_prime(l):
    used = []
    while True:
        a = random.randint(2 ** (l - 1) + 1, 2 ** l - 1)
        if a in used:
            continue
```

```
        if miller_rabin_test(a, 8):
            return a


def main():
    l = int(input("Введите количество бит простого p: "))

    # Генерация Открытый параметр p
    p = get_prime(l)
    while not miller_rabin_test((p - 1) // 2, 8):
        p = get_prime(l)
    print("Открытый параметр p:", p)

    # Генерация Открытый параметр q
    while True:
        g = random.randint(0, p)
        if pow(g, p - 1, p) % p == 1 and miller_rabin_test(g, 8):
            break
    print("Открытый параметр g:", g)


    # Генерация: случайное натуральное число a — закрытый ключ Алисы
    a = random.randint(0, p)
    print("Закрытый ключ a:", a)

    # Генерация: случайное натуральное число b — закрытый ключ Боба
    b = random.randint(0, p)
    print("Закрытый ключ b:", b)

    #Вычисляется открытый ключ Алисы, передается Бобу
    Alice = pow(g, a, p)
    print("Открытый ключ Алисы, который передается Бобу: ", Alice)

    # Вычисляется открытый ключ Боба, передается Алисе
    Bob = pow(g, b, p)
    print("Открытый ключ Боба, который передается Алисе: ", Bob)

    K_alice = pow(Bob, a, p)
    print("Вычисляется общий секретный ключ K на стороне Алисы: ",
K_alice)


    K_bob = pow(Alice, b, p)
    print("Вычисляется общий секретный ключ K на стороне Боба: ",
K_bob)

    print(K_alice == K_bob)


if __name__ == "__main__":
    main()
```