Министерство образования Республики Беларусь
Белорусский государственный университет информатики и радиоэлектроники
Кафедра информатики

ОТЧЕТ
по лабораторной работе №7

Криптография с использованием эллиптических кривых

Выполнил:
студент гр. 653501
Шинкевич Г. С.

Проверил:
Артемьев В. С.

Минск 2019

ЗАДАНИЕ:

Реализовать схему шифрования (дешифрования) для аналога алгоритма Диффи-Хеллмана на основе эллиптических кривых.

ECDH

Криптографический протокол, позволяющий двум сторонам, имеющим пары открытый/закрытый ключ на эллиптических кривых, получить общий секретный ключ, используя незащищённый от прослушивания канал связи. Этот секретный ключ может быть использован как для шифрования дальнейшего обмена, так и для формирования нового ключа, который затем может использоваться для последующего обмена информацией с помощью алгоритмов симметричного шифрования.

Параметры алгоритма ECDH

- Простое **p**, задающее размер конечного поля
- Коэффициенты **a** и **b** уравнения эллиптической кривой
- Базовая точка **G**, генерирующая подгруппу
- Порядок **n** подгруппы
- Закрытый ключ — это случайное целое **d**, из промежутка (0, n)
- Открытый ключ — это точка **H** = d * G

Алгоритм ECDH

Пусть существуют два абонента: Алиса и Боб. Предположим, Алиса хочет создать общий секретный ключ с Бобом, но единственный доступный между ними канал может быть подслушан третьей стороной.

1. Сначала Алиса и Боб генерируют собственные закрытые и открытые ключи. У Алисы есть закрытый ключ $\mathbf{d_A}$ и открытый ключ $\mathbf{H_A} = d_A * G$, у Боба есть ключи $\mathbf{d_B}$ и $\mathbf{H_B} = d_B * G$.

2. Алиса и Боб обмениваются открытыми ключами $\mathbf{H_A}$ и $\mathbf{H_B}$ по незащищённому каналу. Посредник перехватывает $\mathbf{H_A}$ и $\mathbf{H_B}$, но не может определить ни $\mathbf{d_A}$, ни $\mathbf{d_B}$, не решив задачу дискретного логарифмирования.

3. Алиса вычисляет **S** = d_A * H_B (с помощью собственного закрытого ключа и открытого ключа Боба), а Боб вычисляет **S** = d_B * H_A (с помощью собственного закрытого ключа и открытого ключа Алисы). **S** одинаков для Алисы и Боба.

Посреднику известны только H_A и H_B и он не сможет найти общий секретный ключ **S**.

В дальнейшем, общий секретный ключ **S** может использоваться, например, для симметричного шифрования.

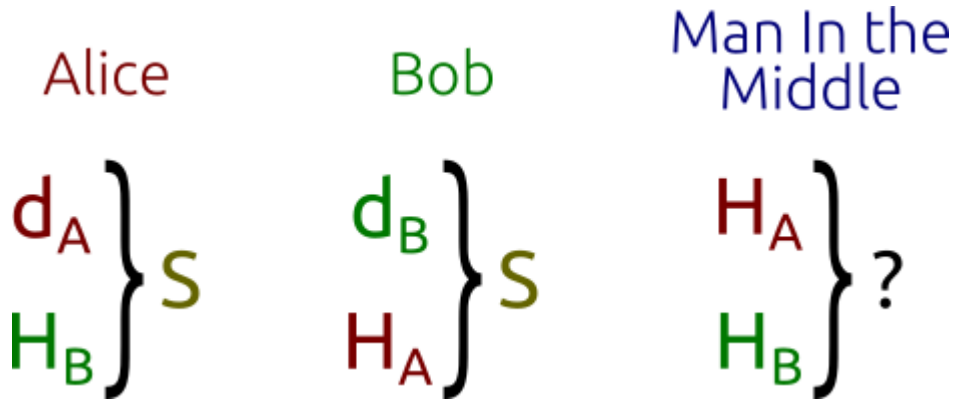

Рисунок 1. Протокол Диффи-Хеллмана

ДЕМОНСТРАЦИЯ РАБОТЫ

```
First private key: ecea97a9ec1e16f7fe45f8315a0279ed992e7dee4022ee91ad2c8b78bf4f3870
First public key: (x=787a24e6daff8020bd317acbac95dc3f43241a08c8e5b21bdc528fc29334298,
  y=216b37fdf874ea126f7f797eb94e69f5a555ff9cfcbd55b4c217dbf7b58c0fa2)
Second private key: b070f001d21d5c5017a74681743fa616412c23b722c017ed89d181018d37139c
Second public key: (x=7431dade89741d0491d12289eec09223a60771def6d480f2d23777b2f0532ba5,
  y=d032459bc9a72410cdbaa815325838f64b7eda19833ec9526a372c48cd9edbb0)
Shared key: b64bc92b28c3576a89fa1fd42a0b8a74228a13d82b967255641023295c641757
Initial text: BSUIR135 (4253554952313335)
Encrypted text: 15404DF76647899A5A5FF9DB333C51BC
Decrypted text: BSUIR135 (4253554952313335)
```

ВЫВОД

В результате лабораторной работы была написана программа для генерации ключей с помощью схемы Диффи-Хеллмана на эллиптических кривых.

ЛИСТИНГ ПРОГРАММЫ

```
import random
import des


class Point:
    def __init__(self, a, b, p, n, x, y):
        self.a = a
        self.b = b
        self.p = p
        self.n = n
        self.x = x
        self.y = y

    def __add__(self, other):
        if (self.x, self.y) == (0, 0):
            return other
        if (other.x, other.y) == (0, 0):
            return self
        if self.x == other.x and (self.y != other.y or self.y == 0):
            return Point(self.a, self.b, self.p, self.n, 0, 0)
        if self.x == other.x and self.y == other.y:
            m = (3 * self.x ** 2 + self.a) * multiplicative_inverse(2 * self.y,
self.p)
        else:
            m = (self.y - other.y) * multiplicative_inverse(self.x - other.x,
self.p)

        x_r = (m ** 2 - self.x - other.x) % self.p
        y_r = (m * (self.x - x_r) - self.y) % self.p

        return Point(self.a, self.b, self.p, self.n, x_r, y_r)

    def __mul__(self, number):
        result = Point(self.a, self.b, self.p, self.n, 0, 0)
        addend = self

        for bit in bits(number):
```

```
            if bit == 1:
                result += addend
            addend += addend
        return result


def bits(number):
    while number:
        yield number & 1
        number >>= 1


def get_bezout_coeffs(a, b):
    # ax + by = gcd(a, b)
    x, x_, y, y_ = 1, 0, 0, 1
    while b:
        q = a // b
        a, b = b, a % b
        x, x_ = x_, x - x_ * q
        y, y_ = y_, y - y_ * q
    return x, y


def multiplicative_inverse(a, b):
    x, y = get_bezout_coeffs(a, b)
    if x < 0:
        return b + x
    return x


def generate_keys(n, G):
    d = random.randint(1, n - 1)
    H = G * d
    return d, H


if __name__ == '__main__':
    p = 0xfffffffffffffffffffffffffffffffffffffffffffffffffffffffefffffc2f
    a = 0x0
```

```
    b = 0x7
    n =
0xfffffffffffffffffffffffffffffffebaaedce6af48a03bbfd25e8cd0364141
    x =
0x79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f
81798
    y =
0x483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb1
0d4b8
    G = Point(a, b, p, n, x, y)

    d1, H1 = generate_keys(n, G)
    print(f"First private key: {d1:x}")
    print(f"First public key: (x={H1.x:x}, y={H1.y:x})")

    d2, H2 = generate_keys(n, G)
    print(f"Second private key: {d2:x}")
    print(f"Second public key: (x={H2.x:x}, y={H2.y:x})")

    S1 = H2 * d1
    S2 = H1 * d2

    print(f"Shared key: {S1.x:x}")

    with open("data.txt", 'r') as file:
        t = file.read()
    k = str(S1.x)
    print(f"Initial text: {t} ({des.str_to_hex(t)})")
    encrypted = des.encrypt(k, t)
    print(f"Encrypted text: {des.str_to_hex(encrypted)}")
    decrypted = des.decrypt(k, encrypted)
    print(f"Decrypted text: {decrypted} ({des.str_to_hex(decrypted)})")
```