**Лабораторная работа №3 по теме «Электронно-цифровая подпись Эль-Гамаля»**

Выполнил студент гр.Б8117-02.03.01

Михайлов Денис

**Цель:**

Требуется реализовать электронно-цифровую подпись Эль-Гамаля.

**Теория:**

**Электронно-цифровая подпись методом Эль-Гамаля:**

Технология применения системы электронной цифровой подписи (ЭЦП) предполагает наличие сети абонентов, посылающих друг другу подписанные электронные документы. Для каждого абонента генерируется пара ключей: секретный и открытый. Секретный ключ хранится абонентом в тайне и используется им для формирования ЭЦП. Открытый ключ известен всем другим пользователям и предназначен для проверки ЭЦП получателем подписанного электронного документа. Иначе говоря, открытый ключ является необходимым инструментом, позволяющим проверить подлинность электронного документа и автора подписи. Открытый ключ не позволяет вычислить секретный ключ.

Идея алгоритма цифровой подписи методом Эль-Гамаля основана на том, что для обоснования практической невозможности фальсификации цифровой подписи может быть использована более сложная вычислительная задача, чем разложение на множители большого целого числа, - задача дискретного логарифмирования.

Для того чтобы сгенерировать пару ключей (открытый и закрытый), сначала выбирают некоторое большое простое целое число и большое целое число , причём . Отправитель и получатель подписанного документа используют при вычислениях одинаковые большие целые числа и , которые не являются секретными.

Отправитель выбирает случайное целое число , и вычисляет

Число является открытым ключом, используемым для проверки подписи отправителя. Число открыто передаётся всем потенциальным получателям документов.

Число является секретным ключом отправителя для подписывания документов и должно храниться в секрете.

Для того чтобы подписать сообщение , сначала отправитель хэширует его с помощью хэш-функции в целое число

и генерирует случайное число , такое, что и являются взаимно простыми. Затем отправитель вычисляет целое число

и, применяя расширенный алгоритм Евклида, вычисляет с помощью секретного ключа целое число из уравнения

Пара чисел образует цифровую подпись

проставляемую под документом .

Тройка чисел передаются получателю, в то время как пара чисел держится в секрете.

После приёма подписанного сообщения получатель должен проверить, соответствует ли подпись

сообщению . Для этого получатель сначала вычисляет по принятому сообщению число

т.е. хэширует принятое сообщение .

Затем получатель вычисляет значение

и признаёт сообщение подлинным, если, и только если

Следует отметить, что выполнения каждой подписи по методу Эль-Гамаля требует нового значения , причём это значение должно выбираться случайным образом.

Функцию хэширования мы взяли следующую:

где – символ хэшируемого сообщения, – значение символа в таблице ascii, – битовый размер числа .

Для нахождения простого числа будем использовать тест простоты Миллера-Рабина.

**Результат работы.**

Message = ‘baaqab’

(175, 101, 1)

<baaqab, 26, 0>

Soo~ nii~ce!

В первой строке – наше сообщение.

Во второй строке содержится свободный ключ, содержащий в себе простое число, первообразный корень по модулю и сам свободный ключ соответственно.

В третьей строке содержится подпись сообщения с самим сообщением.

В последней строке выводится результат проверки подписи. Как видно, результат на проверку положительный.

**Код программы.**

from random import randint

def Prime\_numb(k=300):              #Calculated random Prime number

    string = '1'

    for \_ in range(k-2, -1, -1):

        string += str(randint(0, 1))

    p = int(string, 2)

    if p % 2 != 0:

        p += 1

    p += 1

    while(True):

        if MR(p, k) and MR(2\*p+1, k):

            return 2\*p+1

        p += 2

def MR(numb, k=300):                #The test of Miller-Rabin on simplicity

    d, s = factor(numb-1)

    if numb <= 1:

        return False

    if numb <= 3:

        return True

    if numb % 2 == 0:

        return False

    for \_ in range(k):

        a = randint(2, numb-2 )

        x = Expon(a, d, numb)

        if not (x == 1 or x == numb - 1):

            for j in range(s-1):

                x = Expon(a, 2\*\*(j+1) \* d, numb)

                if x == 1: return False

                if x == numb-1: break

        else: return True

    return True

def factor(numb):

    n = numb

    count = 0

    while(True):

        if n % 2 == 1: return int(n), count

        n /= 2

        count += 1

def Expon(value, degree, mod):      #Exponentiation modulus

    c = 1

    for \_ in range(degree):

        c = c \* value % mod

    return c

def gcd(a, b):                      #Greatest Common Divisor

    while(b != 0):

        a %= b

        a, b = b, a

    return a

def isMutuallyPrime(p):     #Function for finding a mutually Prime number with p

    k = randint(2, p-1)

    while True:

        if k == p:

            k = randint(2, p-1)

        if gcd(k, p) == 1:

            return k

        k += 1

def euler(n):                       #The function of Euler

    r = n

    i = 2

    while i\*i <= n:

        if n % i == 0:

            while n % i == 0:

                n //= i

            r -= r//i

        else:

            i += 1

    if n > 1:

        r -= r//n

    return r

def hashing(message, N = 4):   #division hashing method

    m = 0

    for char in message:

        m += ord(char)

    return m % N

def isQ(mod):                       #Calculating the primitive root modulus p

    count = 100

    k = euler(mod)

    while(True):

        if Expon(count, k, mod) == 1 and count != mod:

            return count

        count += 1

def extended\_gcd(a, b):

    if a == 0: return b, 0, 1

    rezult, q1, q2 = extended\_gcd(b % a, a)

    t = q2 - (b // a) \* q1

    s = q1

    return rezult, t, s

M = 'baaqab'        #Message

p = Prime\_numb(7)    #A Prime number that (p-1)/2 is also a Prime number (The Number Of Sophie Germain)

g = isQ(p)     #Primitive root modulus p (Calculated using the "isQ" function)

k = isMutuallyPrime(p-1) #Find a mutually Prime number with p

m = hashing(M)           #Hash the message

r = Expon(g, k, p)

x = randint(2, p-1)     #Private key

y = Expon(g, x, p)      #Free key

# (p, g, y) - free key

print('({}, {}, {})'.format(p, g, y))

s = (m - x \* r) \* extended\_gcd(k, p - 1)[1] % (p - 1)

print('<{}, {}, {}>'.format(M, r, s))

#The message signature will look like this: <M, r, s>.

#You should check the authenticity of the signature.

#We need check the comparison: y^r \* r^s === g^m   (mod p)

left = Expon(y, r, p) \* Expon(r, s, p) % p

right = Expon(g, m, p)

if left == right:

    print('Soo~ nii~ce!')