Задание

Реализация процедур, составляющих протокол подписи Эль-Гамаля с сокращённой длинной параметров, средствами языка Python 3.7.

Implementation of the procedures that make up the ElGamal signature protocol with shortened parameter lengths using Python 3.7.

Основные теоретические положения

Криптосистема Эль-Гамаля является асимметричной с открытым ключом, основанная на трудности вычисления дискретных логарифмов в конечном поле. Криптосистема включает в себя алгоритм шифрования и алгоритм цифровой подписи. Способ открытого шифрования Эль-Гамаля включает в себя составной частью систему открытого распределения ключей Диффи-Хеллмана.

Схема алгоритма:

Генерация ключей

1. Генерируется случайное простое число p;
2. Выбирается целое число g — первообразный корень g по модулю p ($g < p$). Числа p и g - *параметрами домена* (т.е. открытые параметры системы, используемые одновременно многими пользователями);
3. Выбирается случайное целое число, взаимно простое с $(p - 1)$, x, такое, что $1 < x < p - 1$;
4. Вычисляется $y \equiv g^x mod\ p$;
5. Открытым ключом является *тройка чисел* (p, g, y), закрытым (секретным) ключом — число x держится в секрете:

   - Число y является *открытым ключом*, используемым для проверки подписи отправителя. Число y открыто передаётся всем потенциальным получателям документов.
   - Число x является *секретным ключом отправителя* для подписывания документов и должно храниться в секрете.

В качестве односторонней функции может быть выбрана хэш-функция. Хэш-функция – функция, осуществляющая преобразование массива входных данных произвольной длины в (выходную) битовую строку установленной длины, выполняемое определённым алгоритмом.

Формирование подписи отправителем

Вычисляется хэш сообщения $m = H(M)$, где M – сообщение. При этом должно выполнять условие $1 < m < (p-1)$;

1. Выбирается случайное число k (это рандомизатор, держится в секрете) из интервала $(1;\ p-1)$, взаимно простое с $p-1$, и вычисляется:

   $$R \equiv g^k mod\ p.$$

   Находится число S из уравнения $m \equiv Rx + kS\ mod(p-1)$ т.е.

   $$S \equiv (m - Rx)k^{-1} mod\ (p-1).$$

2. Подписью сообщения M является пара (R, S). Получателю отправляется тройка (M, R, S) в то время как пара чисел (x, k) держится в секрете. Особенность данной электронной цифровой подписи является то, что не допускается использовать одно и то же значение k для формирования подписи для двух разных сообщений, поскольку это делает возможным вычисление секретного ключа. Использованные значения k должны храниться в секрете, обычно после выработки подписи они уничтожаются.

Проверка подписи получателем

Зная открытый ключ (p, g, y), подпись (R, S) сообщения M проверяется следующим образом:

1. Проверяется выполнимость условий: $0 < R < p$ и $0 < S < p-1$. Если хотя бы одно из них не выполнено, то подпись не прошла проверку;
2. Вычисляется хэш сообщения $m = H(M)$;
3. Вычисляется $D1 \equiv g^m (mod\ p)$ и $D2 \equiv y^R R^S (mod\ p)$;
4. Подпись принимается только при условии, что: $D1 = D2$ подпись верна, сообщение не было подделано. Это уравнение получается путём постановки в уравнение проверки подписи значения $R = g^m mod\ p$:

$$g^m \equiv y^R R^S mod\ p.$$

Математическое обоснование корректности схемы подписи

$$y^R R^S mod\ p \equiv g^{Rx} g^{kS} (mod\ p) \equiv g^{Rx+kS} (mod\ p) \equiv g^m (mod\ p).$$

Учитывая, что g – первообразный корень по модулю p, $g^{\varphi(p)} \equiv$ $\equiv 1(mod\ m), \varphi(p) = p - 1$, можем сделать вывод, что отправитель сообщения M обладает именно этим секретным ключом x и подписал именно этот документ.

Схема Эль-Гамаля с сокращённой длиной параметров R и S

Уравнение проверки подписи $g^m \equiv y^R R^S mod\ p$ может выполняться также в случае, когда в качестве g берется число, относящееся к простому показателю q, где $q|p - 1$. Для этого S должно быть вычислено из следующего соотношения:

$$m \equiv Rx + kS\ mod\ q.$$

Соотношение проверки подписи $g^m \equiv y^R R^S mod\ p$ в схеме с сокращенной параметром S $(S < q)$ может быть преобразовано в уравнение следующего вида:

$$R \equiv g^{m/S} y^{-R/S} \bmod p.$$

При этом вместо R в степени при y можно использовать значение некоторой хэш-функции от значения R, т.е. $H(R)$. В этом случае уравнение проверки подписи имеет вид $R = g^{m/S} y^{-H(R)/S} \bmod p$. Чтобы проверка была корректной, владелец секретного ключа должен вычислить параметр S из следующего сравнения:

$$m \equiv xH(R) + kS\ mod\ q.$$

Поскольку при проверке подписи не требуется выполнять никаких вычислений с использованием параметра R, то проверка подписи может быть осуществлена в соответствии с уравнением:

$$H(R) = H\left(g^{m/S} y^{-H(R)/S} \bmod p\right).$$

В этом случае нет необходимости представлять проверяющему значение R, имеющее сравнительно большую длину. Достаточно для проверки

представить значение $H(R)$, где размер значения хеш-функции равен, например, 160 бит. Этим достигается существенное сокращение длины подписи.

Сокращение длины подписи не уменьшает стойкость системы ЭЦП, поскольку сложность задачи дискретного логарифмирования не изменяется, т.е. вычисления ведутся по модулю исходного размера.

В качестве хэш-функции $H(R)$ можно взять следующую $H(R) = R\ mod\ q$, где q – показатель, используемый при сокращении параметра S. Тогда приходим к следующему уравнению проверки подписи:

$$R' = (g^{m/S} y^{-R'/S} mod\ p) mod\ q,$$

где (R', S') есть подписи к сообщению M, а параметр R' вычисляется после выбора случайного числа k в соответствии с формулой $R' = (g^k mod\ p) mod\ q$.

Сравнение, используемое для вычисления параметра S, имеет вид: $M = Rx' + kS\ mod\ q$.

Разработка программы

Программа разрабатывалась для операционной системы Windows.

Для создания программы, реализующую функционал шифрования/дешифрования текста, был выбран язык *Python 3.7.6*. Пример работы программы представлен на рисунках 1-4. Код программ, генерирующих ключи для формирования и проверки ЭЦП представлены в приложении А.

На рисунке 1 показан результат выполнения программы – опции меню. Программа позволяет пользователю выбирать режим.

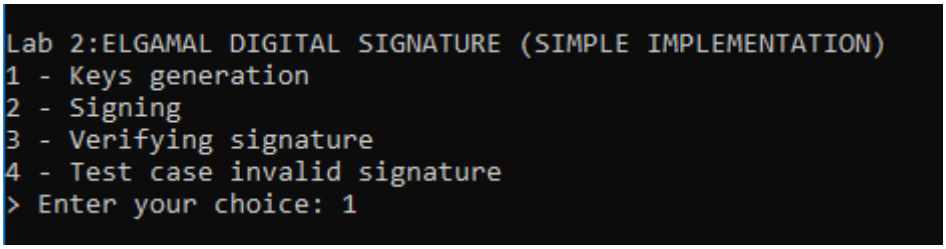

Рисунок 1 – Опции меню

На рисунке 2 показан результат выполнения генерации ключей. Программа генерирует случайное простое число p в соответствии с длиной в битах, указанной пользователем. В то же время генерируются значения g, x, y, которые удовлетворяют алгоритму генерации ключей Эль-Гальмана.

Рисунок 2 – Генерация ключей

После получения параметров домена (p, g), закрытого ключа x, открытого ключа y, программа переходит к формированию подписи отправителем. Для выполнения хеш-функции, необходимой для работы подписи, реализовать выбранную хеш-функцию SHA-256 (256 бит), после ввода сообщения для подписи. Подпись создаётся как пара чисел (R, S) с длиной бит, показанной на рисунке 3.

Рисунок 3 – Формирование подписи отправителем

После формирования подписи на экране отображаются комментарии к процессу проверки подписи и заключение о корректности подписи. Если подпись удовлетворяет условию проверки, программа выводит на экран сообщение о том, что подпись верна (Рисунок 4).

Рисунок 4 – Сообщение и его верная подпись

Подпись имеет два параметра (R, S), если передать на проверку измененный полученный параметр, уравнение проверки подписи не выполнится, это приводит к разным результатам вычислений $D1$ и $D2$, подпись не прошла проверку ($D1 \neq D2$ - не удовлетворяет условию уравнения проверки подписи). Следовательно, на рисунке 4 показано на экране: подпись неверна!

Рисунок 5 – Сообщение с неверной подписью

Ниже представлена таблица 1 с описанием используемых в программе функций.

Таблица 1 – Разработанные функции

№	Определение функции	Описание функции
1	`def MillerRabin(n)`	Функция Миллера — Рабина часто используется в криптографии для получения больших случайных простых чисел.
2	`def genprimeBits(k):`	Функция для генерации простых чисел с заданным количеством битов.
3	`def mod(a,p)`	Функция для приведения a по модулю p
4	`def fast_pow(a,w,n)`	Функция для генерирования g - число относящееся к q как к показателю

Заключение

В данной лабораторной работе были реализованы операции протокола формирования и проверки ЭЦП Эль-Гамаля с сокращённой длинной параметров.

Криптографическая стойкость данной схемы цифровой подписи основывается на сложности решения задачи дискретного логарифмирования, а также на стойкости используемой хэш-функции.

Программа, разработанная на языке Python 3.7, реализует процедуры схемы электронной подписи сообщений Эль-Гамаля. Результаты тестирования разработанной программы подтвердили правильность ее вычислений. В ходе работы была доказана корректность схемы подписи.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. «Алгоритмы электронной цифровой подписи». URL: http://crypto-r.narod.ru/glava6/glava6_3.html (Дата обращения: 08.12.2020). – Текст электронный.

2. Молдовян Н.А. Практикум по криптосистемам с открытым ключом, БХВ-Петербург, 2007-304с.

ПРИЛОЖЕНИЕ А

Program KP_Lab 2.py

```
##!/usr/bin/env python3
##Реализация процедур, составляющих протокол подписи Эль-Гамаля с сокращённой длинной параметров.
import Crypto.Util.number as num
from Crypto import Random
from Crypto.Util import number
import random
import sympy
import hashlib
import sys
from hashlib import sha256
from binascii import hexlify, unhexlify


def MillerRabin(n):
        if n!=int(n):
            return False
        n=int(n)
        #Miller-Rabin test for prime
        if n==0 or n==1 or n==4 or n==6 or n==8 or n==9:
            return False
        if n==2 or n==3 or n==5 or n==7:
            return True
        s = 0
        d = n-1
        while d%2==0:
            d>>=1
            s+=1
        assert(2**s * d == n-1)

        def trial_composite(a):
            if pow(a, d, n) == 1:
                return False
            for i in range(s):
                if pow(a, 2**i * d, n) == n-1:
                    return False
            return True
        for i in range(8):#number of trials
            a = random.randrange(2, n)
            if trial_composite(a):
                return False
        return True

def genprimeBits(k):
    x = ""
    k = int(k)
    for y in range(k):
        x = x + "1"
    y = "1"
    for z in range(k-1):
        y = y + "0"
    x = int(x,2)
    y = int(y,2)
    p = 0
    while True:
        p = random.randint(y,x)
        if MillerRabin(p):
            break
    return p
```

```
def main_menu():
    while True:
        print("\nLab 2:ELGAMAL DIGITAL SIGNATURE (SIMPLE IMPLEMENTATION)")
        print("1 - Keys generation")
        print("2 - Signing")
        print("3 - Verifying signature")
        print("4 - Exit")
        s = int(input('> Enter your choice: '))
        if (s==1):
            print("\nKEYS GENERATION:")
            l = int(input("Enter length bits of prime p -> "))

            while 1:
                q= genprimeBits(l)
                p=2*q+1 #выбор показатель q, q|p-1
                if num.isPrime(p):
                    break

            while 1:
                g=num.getRandomRange(3,p)
                safe=1
                if pow(g,2,p)==1:
                    safe==0
                if safe and pow(g,q,p)==1:
                    safe=0
                if safe and divmod(p-1,g)[1]==0:
                    safe=0
                    # g^(-1) must not divide p-1 because of Khadir's attack
                ginv=num.inverse(g,p)
                if safe and divmod(p-1,ginv)[1]==0:
                    safe=0
                if safe:
                    break
            while(num.GCD(g,p-1)!=1):
                g=num.getRandomRange(3,p)

            print("Prime number p ->", p,"\nBinary p ->","(",bin(p),")")
            print("\nGenerator g ->",g)

            x=num.getRandomRange(1,p-1)
            while (num.GCD(x,p-1)!=1):
                x=num.getRandomRange(1,p-1)
            print("\nPrivate key x ->",x)
            y=pow(g,x,p)
            print("\nCalculating y=g^x mod p -> y=",y)
            print("\nCombo public key (p,g,y)->",p,g,y)
        elif (s==2):
            print("\nSIGNING:")
            input_message = input("Message: ")
            inputbytes = str.encode(input_message)
            while 1:
                k=num.getRandomRange(1,p-2)#k-random number in range(1,p-2), HOD(k,p-1)=1
                if (num.GCD(k,p-1)==1):
                    break
            #length of hash=256 bits
            h = hashlib.sha256(input_message.encode('utf-8')).hexdigest()
            mes = int(h,16)
            print("\nHashed message (length bits M must < length bits p) ->
(",mes.bit_length(),"bits)",mes)
            R=pow(g,k,p)
            print("\nParameter signature R (",R.bit_length(),"bits) -> ",R)
            t=num.inverse(k,q)
            S=t*(mes-R*x)%(q)
            print("\nParameter signature S (",S.bit_length(),"bits)-> ",S)
            print("\nSinging signature (R,S).... -> (",R, S,")")
```

```
            print("\nMessage is signed! [M,R,S] -> [",input_message,"(",R,S,")]")
            print("\nSecret parameter k ->",k)
        elif (s==3):
            print("\nVERIFYING A SIGNATURE:")
            if (R>p) or (S>(p-1)) or (num.GCD(g,(p-1))!=1) :
                print ("\nWrong Signature! Invalid parameters R or S")
            D1=pow(g,mes,p)
            D2=(pow(y,R,p)*pow(R,S,p))%p
            print ("\nD1=g^m mod p->",D1)
            print("\nD2=y^R*R^Smod p->",D2)
            if (D1==D2):
                print("\nCorrect Signature!")
            else:
                print("\nWrong signature! Invalid parameters R or S")

        elif (s==4):
            exit()


main_menu()
```