

Задание

Реализация процедур, составляющих протокол подписи Эль-Гамала с сокращённой длиной 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-Гамала 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-Гамала 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 $y \equiv g^x \bmod p$;
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 выполняться 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:

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

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

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

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 $D1 \equiv g^m \bmod p$ и $D2 \equiv y^R R^S \bmod p$;
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 $R = g^m \bmod p$:

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

Математическое обоснование корректности схемы подписи

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

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

Схема Эль-Гамала с сокращённой длиной параметров R и S

Уравнение проверки подписи $g^m \equiv y^R R^S \bmod p$ 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:

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

Соотношение проверки подписи $g^m \equiv y^R R^S \bmod p$ в схеме с сокращённым 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:

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

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

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

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 $H(R) = R \bmod q$, где q – показатель, используемый при сокращении параметра S . Тогда приходим к следующему уравнению проверки подписи:

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

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

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

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.

```
Lab 2:ELGAMAL DIGITAL SIGNATURE (SIMPLE IMPLEMENTATION)
1 - Keys generation
2 - Signing
3 - Verifying signature
4 - Test case invalid signature
> Enter your choice: 1
```

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.

```
KEYS GENERATION:
Enter length bits of prime p -> 1024
Prime number p -> 150390566074511848145520060915796010565614398756051844641968198717139282564932250998161217230241125456539938928398250
6780064174081882001976897858148528898256910871673064224354853171443022020607376183524303377053182380248693981717463616313198035304700251
985521996726465776069364695631207715596335179568284697193
Binary p -> ( 0b110101100010100111001100011000110110101010001101110101011101110100111011100001110111000101110111110011000001
111000111010010011111000110010010111011000010001001111010100010001100110101010001000110011010101000101001101111100101011101010
1110111000011101111110001010000110100001000111010111011101010100111011101000001010001100100100100110011101000001111001110
111000010010101100110000100110110010000010010011010011011111010101110010000110111001010001010101100011100000010100100100010001
0010011100000110000001010001110111100100100100001001001100111000010010011010111001100010001101011100111001111111001010010011100
101110110110100101110001011100010110101001001100010110001000011101110111010111010010101001010011100101100001010010001000010111
0000010000001011000110011111111001001000100011101001110000100100111010111001000110111001100010001110001101100001011111101101001001
0001110101010111000001101011011100101101001010001011000111011110111110100111001101001 )

Generator g -> 2729381326356880125857705912932190075103291250662735373230561262371290470319871132253502210021489456645883487211257320253
688390459459334268342159042887974794228855256212591203467568487262444990766185063561097261150170743632170271145796175170962992520223130
2169289424757338044601948836394948878951037541605760

Private key x -> 102458714415784227221243596616453785646689011120888029574466666239933105676885273313810366206800296448572794380904275
51394103750784296108868253052628071216950069906257403698754136893541387372027412778330695617825312437443930799811105519965923693166156
75578995712635100947869428771965759210040765712152295823

Calculating y=g*x mod p -> y= 844814091684006799597710991306888085259965406377362643624586489179120967454079207884385073332736034927688
848908210647437612331712147488522489606021700634388139554400201599540314517674749653794421963261601765770747679836241881124756320994355
46141272490510681077506283113655974244599551903681972543735827820739

Combo public key (p,g,y)-> 150390566074511848145520060915796010565614398756051844641968198717139282564932250998161217230241125456539938
928398150678064174081882001976897858148528898256910871673064224354853171443022020607376183524303377053182380248693981717463616313198035
364780251905521900726405776869364605631207715596335179568284697193 272938132635688012585770591293219007510329125066273537323056126237129
047031987113225350221002148945664588348721125732025368839045945933426834215904288797479422885525621259120346756848726244499076618506356
109726115017074363217827114579617517096299252022313821692890242757338044601940836394948870951037543605760 84481409168400679959771099130
688808525996540637736264362458648917912096745407920788438507333273603492768884890821064743761233171214748852248960802170863438813955440
820159954031451767474965379442196326160176577074767983624108112475632099435546141272490518081077506283113655974244599551903681972543735
872820739
```

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.

```

# - test case invalid signature
> Enter your choice: 2

SIGNING:
Message: leti

Hashed message -> ( 255 bits) 38244815803731895272928439296714526384286832899952923915548946886797775287279

Parameter signature R=g^k mod p ( 1023 bits) -> 69701311563886324274351059674884594666121691055039812685049449557443101954766911358117
404496332638216865602491169059641885847628630107864541307785253667042671337152468338426901445898712280067599767473982716800272410465900
277757808255365142590842566648042517849883602973456072434002269177701375924972785164865

Parameter signature S=(M-Rx)k^(-1) mod p ( 80 bits)-> 618229967093811322158632

Signing signature (R,S).... -> ( 697013115638863242743510596748845946661216910550398126850494495574431019547669113581174044963326382168
656024911690596418858476286301078645413077852536670426713371524683384269014458987122800675997674739827168002724104659002777578082
42590842566648042517849883602973456072434002269177701375924972785164865 618229967093811322158632 )

Message is signed! [M,R,S] -> [ leti ( 697013115638863242743510596740045946661216910559398126850494495574431019547669113581174044963326
382168656024911690596418858476286301078645413077852536670426713371524683384269014458987122800675997674739827168002724104659002777578082
55365142590842566648042517849883602973456072434002269177701375924972785164865 618229967093811322158632 ) ]

Secret parameter k -> 77915641542870369419620533698716796240865348781307732622863693771057094824367644036323730974409633974306483372249
226812345731243861372961132458918148379358871394529235874301659159911301887779580834624105197891772121833683265031548473440145588788335
564836241271755729972304130187957602385885579001098200575165

```

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).

```

> Enter your choice: 3

VERIFYING A SIGNATURE:

Wrong Signature! Invalid parameters R or S.

D1=g^m mod p-> 574288323015906013455672257614196777029853259966446954799528766087715712176486284821897723446755119206704456620082774926
568135171695877591223998168348117475357711967222321875448918021982963537885247356104794585856947839864298354620414564048978337771793519
55810509358772113268298016298709138925673290496531178

D2=y^R*R^Sm mod p-> 574288323015906013455672257614196777029853259966446954799528766087715712176486284821897723446755119206704456620082774
920560135171695877591223998168348117475357711967222321875448918021982963537885247356104794585856947839864298354620414564048978337771793
51955810509358772113268298016298709138925673290496531178

Correct Signature!

```

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!

```
4.TEST CASE INVALID SIGNATURE

New k' -> 89874022558209148548528330231499573012278180392257796766902905807269710115565

Parameter signature (changed) R'=g^k' mod p ( 255 bits) -> 545649052611605597370343480244808518138112492255764260179043
98906193728577248

Signing signature (R',S)... -> ( 54564905261160559737034348024480851813811249225576426017904398906193728577248 58458674
924431242900926824440917981437161442089410157047915872120565504112 )

Message is signed! [M,R',S] -> [ leti ( 54564905261160559737034348024480851813811249225576426017904398906193728577248 58
458674924431242900926824440917981437161442089410157047915872120565504112 )]

D1'=g^m mod p-> 40663432697884940721702342319591269751347961921519720417031754437037580145522

D2'=y^R'*R'^S mod p-> 26987733044677404066623595927551501857525296872474470351338679741295386048401

Wrong signature! Invalid parameters R or S
```

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 <i>a</i> по модулю <i>p</i>
4	def fast_pow(a,w,n)	Функция для генерирования <i>g</i> - число относящееся к <i>q</i> как к показателю

Заключение

В данной лабораторной работе были реализованы операции протокола формирования и проверки ЭЦП Эль-Гамала с сокращённой длиной 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-Гамала. 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-Гамала с сокращённой длиной
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("\nSigning 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()

```