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

**Схемы аутентификации**

ОТЧЁТ

ПО ДИСЦИПЛИНЕ

«КРИПТОГРАФИЧЕСКИЕ ПРОТОКОЛЫ»

студента 5 курса 531 группы

специальности 10.05.01 Компьютерная безопасность

факультета компьютерных наук и информационных технологий

Серебрякова Алексея Владимировича

|  |  |  |
| --- | --- | --- |
| Преподаватель  аспирант | \_\_\_\_\_\_\_\_\_\_ | Р. А. Фарахутдинов |
|  | подпись, дата |  |

Саратов 2023

**1 Постановка задачи**

Цель работы:

* Изучение протокола аутентификации Шнорра (Схемы Шнорра) и его программная реализация.

Задачи работы:

* Изучить протокол Шнорра, его сильные и слабые стороны;
* Привести программную реализацию протокола.

**2 Теоретические сведения**

Схемы аутентификации и электронной подписи — одни из наиболее важных и распространенных типов криптографических протоколов, которые обеспечивают целостность информации.

Понять назначение протоколов аутентификации можно легко на следующем примере. Предположим, что у нас есть информационная система, в которой необходимо разграничить доступ к различным данным. Также в данной системе присутствует администратор, который хранит все идентификаторы пользователей с сопоставленным набором прав, с помощью которого происходит разграничение доступа к ресурсам. Одному пользователю может быть одновременно разрешено читать один файл, изменять второй и в то же время закрыт доступ к третьему. В данном примере под обеспечением целостности информации понимается предотвращение доступа к системе лиц, не являющихся её пользователями, а также предотвращение доступа пользователей к тем ресурсам, на которые у них нет полномочий. Наиболее распространенный метод разграничения доступа, парольная защита, имеет массу недостатков, поэтому перейдем к криптографической постановке задачи.

Схема Шнорра (англ. schnorr scheme) — одна из наиболее эффективных и теоретически обоснованных схем аутентификации. Безопасность схемы основывается на трудности вычисления дискретных логарифмов. Предложенная Клаусом Шнорром схема является модификацией схем Эль-Гамаля (1985) и Фиата-Шамира (1986), но имеет меньший размер подписи. Схема Шнорра лежит в основе стандарта Республики Беларусь СТБ 1176.2-99 и южнокорейских стандартов KCDSA и EC-KCDSA. Она была покрыта патентом США 4999082, который истек в феврале 2008 года.

В протоколе имеются два участника — Алиса, которая хочет подтвердить свой идентификатор, и Боб, который должен проверить это подтверждение. У Алисы имеется два ключа — , открытый (общедоступный), и — закрытый (приватный) ключ, известный только Алисе. Фактически, Боб должен проверить, что Алиса знает свой закрытый ключ , используя только .

Схема Шнорра — одна из наиболее эффективных среди практических протоколов аутентификации, реализующая данную задачу. Она минимизирует зависимость вычислений, необходимых для создания подписи, от сообщения. В этой схеме основные вычисления могут быть сделаны во время простоя процессора, что позволяет увеличить скорость подписания. Как и DSA, схема Шнорра использует подгруппу порядка в . Также данный метод использует хеш-функцию .

Генерация ключей для схемы подписи Шнорра происходит так же, как и генерация ключей для DSA, кроме того, что не существует никаких ограничений по размерам. Заметим также, что модуль может быть вычислен автономно.

1. Выбирается простое число , которое по длине обычно равняется 1024 битам.
2. Выбирается другое простое число таким, чтобы оно было делителем числа . Или, другими словами, должно выполняться . Размер для числа принято выбирать равным 160 битам.
3. Выбирается число , отличное от 1, такое, что .
4. Пегги выбирает случайное целое число , меньшее .
5. Пегги вычисляет .
6. Общедоступный ключ Пегги - , секретный ключ Пегги - .

Рассмотрим алгоритм работы протокола:

1. Предварительная обработка. Алиса выбирает случайное число , меньшее , и вычисляет . Эти вычисления являются предварительными и могут быть выполнены задолго до появления Боба.
2. Инициирование. Алиса посылает Бобу.
3. Боб выбирает случайное число из диапазона от 0 до и отправляет его Алисе.
4. Алиса вычисляет и посылает Бобу.
5. Подтверждение. Боб проверяет что .

Безопасность алгоритма зависит от параметра . Сложность вскрытия алгоритма примерно равна . Шнорр советует использовать около 72 битов, для и . Для решения задачи дискретного логарифма, в этом случае, требуется по крайней мере шагов известных алгоритмов.

**3 Тестирование программы**

На рисунках 1-3 представлены результаты работы программы, эмулирующей работу протокола аутентификации Шнорра между двумя субъектами (Alice и Bob).

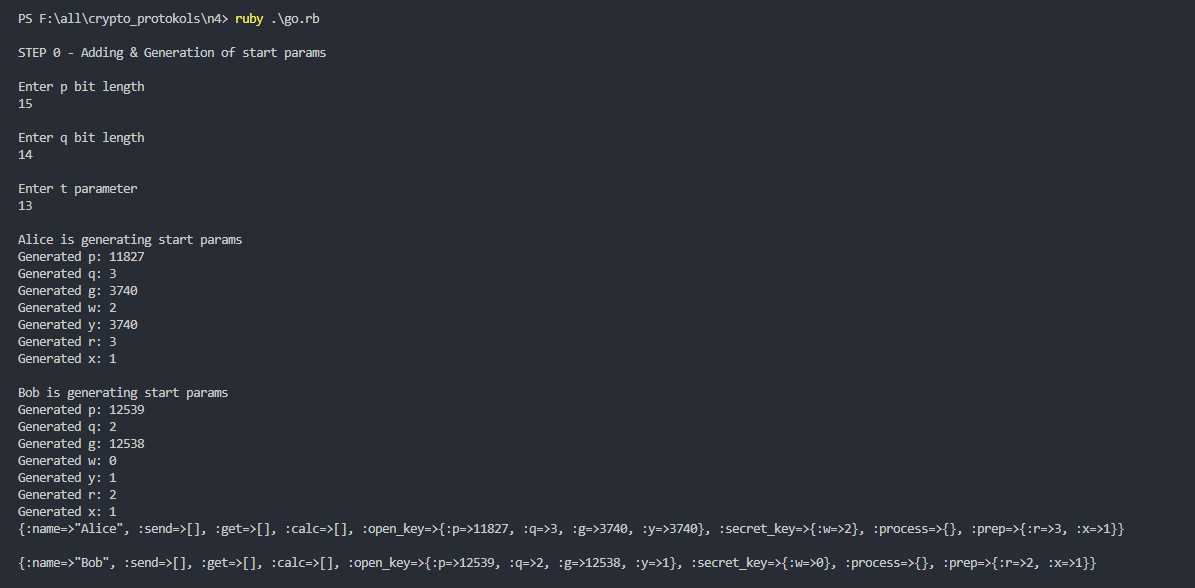


Рисунок 1 - Результат работы программы

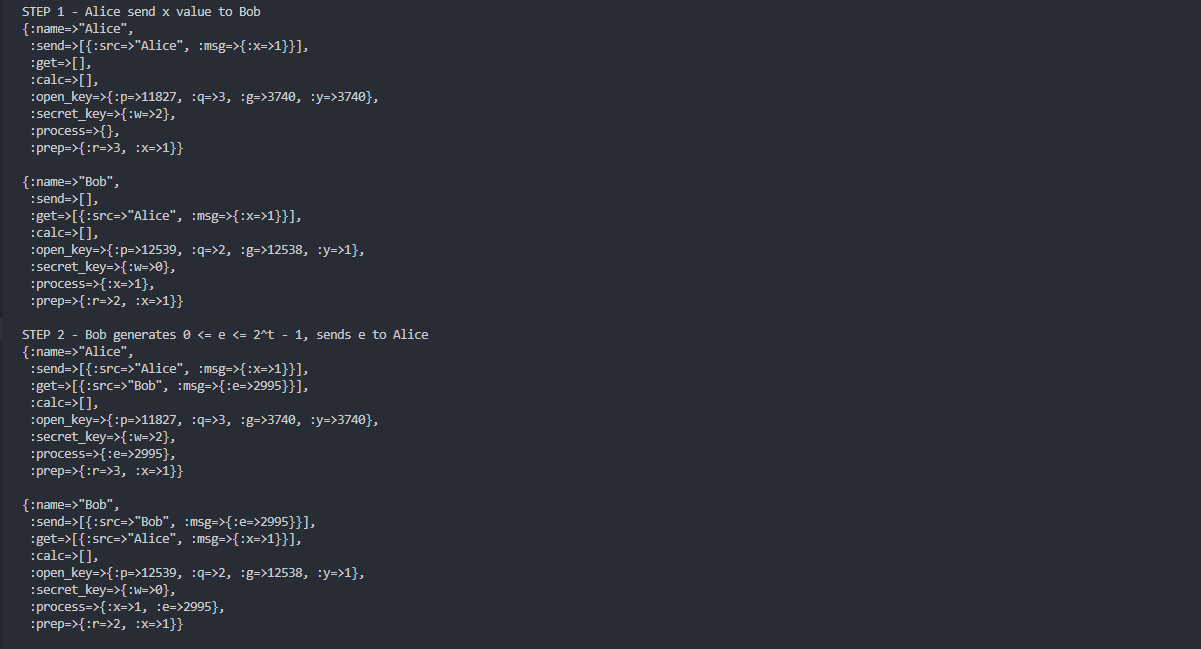


Рисунок 2 - Результат работы программы

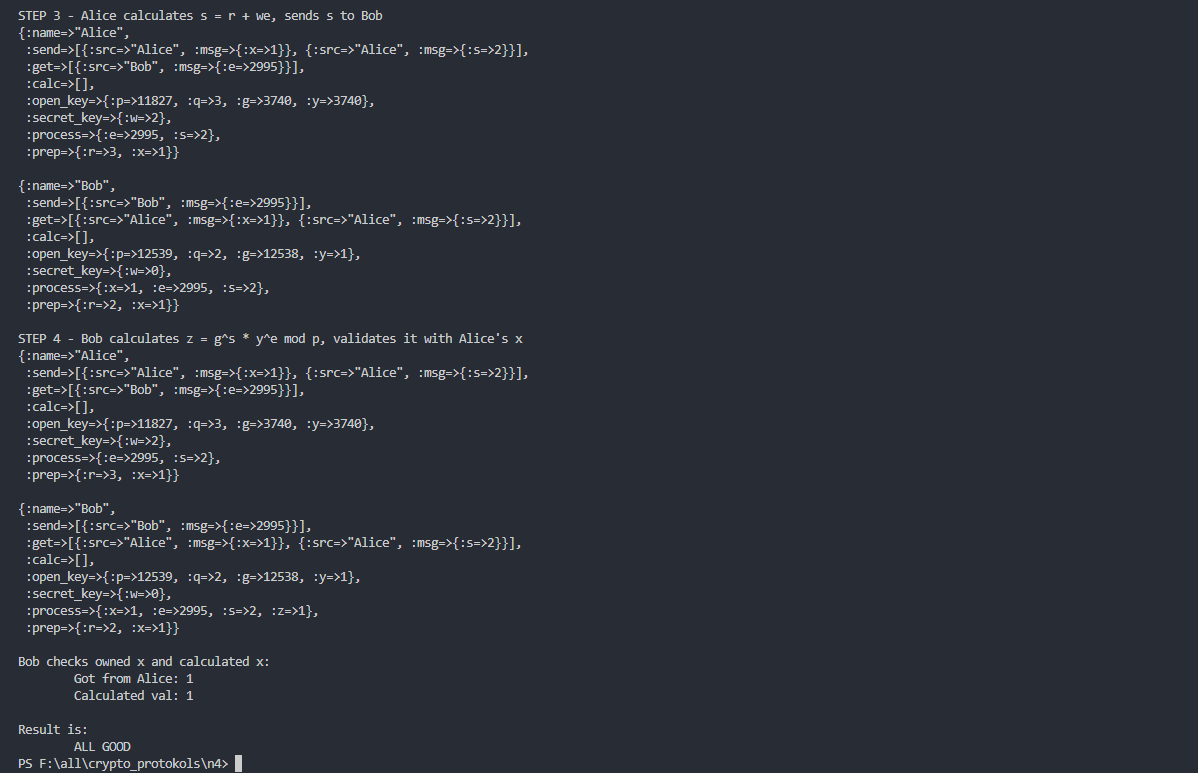


Рисунок 3 - Результат работы программы

**ПРИЛОЖЕНИЕ А**

**Код программы go.rb**

require './steps.rb'

@steps = Steps.new

@steps.step0

@steps.step1

@steps.step2

@steps.step3

@steps.step4

**ПРИЛОЖЕНИЕ Б**

**Код программы keygen.rb**

require 'prime'

require 'openssl'

class Keygen

def initialize

end

def gen\_keys

*# Шаг 1: Генерация простого числа p (битовая длина 1024)*

  p = Prime.each(2\*\*(1024-1), 2\*\*1024 - 1).first

  puts p

*# Шаг 2: Генерация простого числа q (битовый размер 160 битов)*

  q = Prime.each(2\*\*(160-1), 2\*\*160 - 1).first

  puts q

*# Шаг 3: Генерация случайного числа g, такого что g^q = 1 (mod p)*

  g = find\_random\_g(p, q)

  puts g

*# Шаг 4: Генерация случайного числа w < q*

  w = SecureRandom.random\_number(q)

  puts w

*# Шаг 5: Вычисление y = g^(q-w) (mod p)*

  y = g.mod\_pow(q - w, p)

  puts y

*# Шаг 6: Сохранение ключей*

  open\_key = { p: p, q: q, g: g, y: y }

  private\_key = { w: w }

  {open\_key: open\_key, private\_key: private\_key}

end

private

def find\_random\_g(p, q)

  loop do

    g = SecureRandom.random\_number(p-2) + 2

    return g if g.mod\_pow(q, p) == 1

  end

end

end

**ПРИЛОЖЕНИЕ В**

**Код программы methods.rb**

require 'securerandom'

require 'prime'

class Methods

  def initialize(params = {})

    @p\_bit\_length = params.dig(:p\_bit\_length) || 1024

    @q\_bit\_length = params.dig(:q\_bit\_length) || 160

    @t =            params.dig(:t) || 72

    @debug\_mode = params.dig(:debug\_mode) == :all

  end

  def make\_keys(name = 'Client')

    puts "\n#{name} is generating start params"

    p = gen\_large\_p

    puts "Generated p: #{p}" if @debug\_mode

    q = gen\_del\_q(p)

    puts "Generated q: #{q}" if @debug\_mode

    g = gen\_step\_g(p, q)

    puts "Generated g: #{g}" if @debug\_mode

    w = gen\_rand\_w(q)

    puts "Generated w: #{w}" if @debug\_mode

    y = find\_y(g, q, w, p)

    puts "Generated y: #{y}" if @debug\_mode

    keys = {

      open\_key:{

        p: p,

        q: q,

        g: g,

        y: y

      },

      secret\_key:{

        w: w

      },

      process:{

      }

    }

    keys[:prep] = prep\_enum(keys)

    puts "Generated r: #{keys[:prep][:r]}" if @debug\_mode

    puts "Generated x: #{keys[:prep][:x]}" if @debug\_mode

    return keys

  end

  private

  def gen\_large\_p

    loop do

      p = rand(2 \*\* @p\_bit\_length)

      return p if p.prime?

    end

  end

  def gen\_del\_q(p)

    loop do

      q = rand(2..Math.sqrt(p-1).to\_i)

      return q if q.prime? && (p-1) % q == 0

    end

  end

  def gen\_step\_g(p, q)

    loop do

      g = rand(p)

      return g if g != 1 && g.pow(q, p) == 1 && g != 0

    end

  end

  def gen\_rand\_w(q)

    return rand(q)

  end

  def find\_y(g, q, w, p)

    return g.pow(q - w, p)

  end

  def gen\_rand\_r(q)

    return rand(2..q)

  end

  def find\_x(g, r, p)

    return g.pow(r, p)

  end

  def prep\_enum(keys)

    r = gen\_rand\_r(keys[:open\_key][:q])

    x = find\_x(keys[:open\_key][:g], r, keys[:open\_key][:p])

    {

      r: r,

      x: x

    }

  end

end

**ПРИЛОЖЕНИЕ Г**

**Код программы process.rb**

require './keygen.rb'

@keygen = Keygen.new

puts @keygen.gen\_keys

**ПРИЛОЖЕНИЕ Д**

**Код программы steps.rb**

require './methods.rb'

require 'securerandom'

class Steps

  def initialize()

    @alice = nil

    @bob = nil

  end

  def step0

    puts "\nSTEP 0 - Adding & Generation of start params"

    puts "\nEnter p bit length"

    @p\_bit\_length = gets.strip.to\_i

    puts "\nEnter q bit length"

    @q\_bit\_length = gets.strip.to\_i

    puts "\nEnter t parameter"

    @t = gets.strip.to\_i

    params = {

      p\_bit\_length: @p\_bit\_length,

      q\_bit\_length: @q\_bit\_length,

      t: @t,

      debug\_mode: :all

    }

    @methods = Methods.new(params)

    @alice = {name: 'Alice', send:[], get:[], calc:[]}.merge(@methods.make\_keys('Alice'))

    @bob = {name:'Bob', send:[], get:[], calc:[]}.merge(@methods.make\_keys('Bob'))

    pp @alice

    puts "\n"

    pp @bob

  end

  def step1

    puts "\nSTEP 1 - Alice send x value to Bob"

    @alice[:send] << make\_msg(@alice[:name], {x: @alice[:prep][:x]})

    @bob[:get] << @alice[:send].last

    @bob[:process][:x] = @bob[:get].last[:msg][:x]

    pp @alice

    puts "\n"

    pp @bob

  end

  def step2

    puts "\nSTEP 2 - Bob generates 0 <= e <= 2^t - 1, sends e to Alice"

    @bob[:process][:e] = gen\_e

    @bob[:send] << make\_msg(@bob[:name], {e: @bob[:process][:e]})

    @alice[:get] << @bob[:send].last

    @alice[:process][:e] = @alice[:get].last[:msg][:e]

    pp @alice

    puts "\n"

    pp @bob

  end

  def step3

    puts "\nSTEP 3 - Alice calculates s = r + we, sends s to Bob"

    @alice[:process][:s] = find\_s(@alice)

    @alice[:send] << make\_msg(@alice[:name], {s: @alice[:process][:s]})

    @bob[:get] << @alice[:send].last

    @bob[:process][:s] = @bob[:get].last[:msg][:s]

    pp @alice

    puts "\n"

    pp @bob

  end

  def step4

    puts "\nSTEP 4 - Bob calculates z = g^s \* y^e mod p, validates it with Alice's x"

    @bob[:process][:z] = client\_b\_check\_x(@alice, @bob)

    pp @alice

    puts "\n"

    pp @bob

    puts "\nBob checks owned x and calculated x:"

    puts "\tGot from Alice: #{@bob[:process][:x]}"

    puts "\tCalculated val: #{@bob[:process][:z]}"

    puts "\nResult is:"

    puts (@bob[:process][:x] == @bob[:process][:z] ? "\tALL GOOD" : "\tSOMETHING GONE WRONG")

  end

  private

  def make\_msg(src, msg)

    {

*# uid:  SecureRandom.uuid,*

*# time: Time.now,*

      src:  src,

      msg:  msg

    }

  end

  def gen\_e

    return rand(2 \*\* @t)

  end

  def find\_s(client)

    return (client[:prep][:r] + client[:secret\_key][:w] \* client[:process][:e]) % client[:open\_key][:q]

  end

  def client\_b\_check\_x(client\_a, client\_b)

    (client\_a[:open\_key][:g].pow(client\_b[:process][:s], client\_a[:open\_key][:p]) \*

    client\_a[:open\_key][:y].pow(client\_b[:process][:e], client\_a[:open\_key][:p])) % client\_a[:open\_key][:p]

  end

end