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

**Скрытый канал связи**

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 понятия скрытого канала связи и его программная реализация на основе протокола DSA.

Задачи работы:

* Изучить протокол DSA, возможность внедрения в него скрытого канала связи;
* Привести программную реализацию протокола DSA со скрытым каналом связи.

**2 Теоретические сведения**

DSA (англ. Digital Signature Algorithm — алгоритм цифровой подписи) — криптографический алгоритм с использованием закрытого ключа (из пары ключей: <открытый; закрытый>) для создания электронной подписи, но не для шифрования (в отличие от RSA и схемы Эль-Гамаля). Подпись создается секретно (закрытым ключом), но может быть публично проверена (открытым ключом). Это означает, что только один субъект может создать подпись сообщения, но любой может проверить её корректность. Алгоритм основан на вычислительной сложности взятия логарифмов в конечных полях.

Алгоритм был предложен Национальным институтом стандартов и технологий (США) в августе 1991 и является запатентованным (автор патента — David W. Kravitz), НИСТ сделал этот патент доступным для использования без лицензионных отчислений. DSA является частью DSS (англ. Digital Signature Standard — стандарт цифровой подписи), впервые опубликованного 15 декабря 1998 (документ FIPS-186 (англ. Federal Information Processing Standards — федеральные стандарты обработки информации)). Стандарт несколько раз обновлялся, последняя версия FIPS-186-4. (июль 2013).

**Основные параметры схемы.**

Алиса подписывает открытое сообщение , Боб проверяет подпись. Генерируются: — большое простое число с условием ; — большое простое число , такого, что делится на , а именно имеет вид . Выбирается число такое, что его мультипликативный порядок по модулю равен . Для его вычисления можно воспользоваться формулой , где — некоторое произвольное число, такое, что . В большинстве случаев значение удовлетворяет этому требованию. Далее выбирается случайное число и вычисляется . Элементы могут быть общими для группы пользователей, Элемент объявляется открытым ключом, элементы — закрытым ключом Алисы.

Генерация подписи.

1. : , выбирается случайное число ;

2. : {r}, где ;

3. : , где ;

4. : если или , то выбор нового ;

5. : .

Проверка подписи.

6. : , где ;

7. : , где ;

8. : , где ;

9. : , где ;

10. : если , то подпись верна.

**Скрытый канал на основе DSA**

Основные параметры схемы в случае скрытого канала те же, только с добавлением простого числа (отличающегося от параметра в схеме подписи). Это секретный ключ для скрытого канала, известный Алисе и Бобу. Следующая схема позволяет Алисе и Бобу обмениваться в каждой подписи одним битом скрытой информации.

1. Алиса подписывает безобидное сообщение и на шагах 1 и 2 выбирает случайное число так, чтобы параметр подписи являлся квадратичным вычетом по модулю , если она хочет передать скрытый бит 1, или являлся являлся квадратичным невычетом по модулю , если она хочет передать передать скрытый бит 0. Так как числа, являющиеся квадратичными вычетами и н е- вычетами, равновероятны, то добиться результата не сложно.

2. Алиса посылает Бобу подписанное сообщение.

3. Боб проверяет подпись и убеждается в подлинности сообщения. Затем он проверяет, является ли квадратичным вычетом по модулю и восстанав- ливает скрытый бит. Передача описанным выше способом нескольких битов подразумевает подбор такого значения , которое является квадратичным вычетом или невычетом по нескольким модулям .

Скрытый канал использует то обстоятельство, что Алиса может выбирать k для передачи скрытой информации. Чтобы уничтожить скрытый канал связи, необходимо запретить Алисе возможность свободного выбора k. Однако свободный выбор k должен быть запрещен и для всех других лиц, иначе они получат возможность подделать подпись Алисы. Единственным решением в таких обстоятельствах является проведение генерации k вместе с другой стороной, в нашем случае Уолтером, так, чтобы Алиса не могла управлять ни одним битом k, а Уолтер не мог определить ни один бит k. И у Уолтера должна быть возможность проверить, что Алиса использовала именно совместно созданное k.

1. : , где и случайное число ;

2. : , где случайное число ;

3. : , где , далее продолжается основной протокол подписи DSA с шага 2, когда наступает время проверки подписи Уолтером, этот протокол продолжается;

4. : , где ; если , то Уолтер знает, что для подписи m использовалось k.

После этапа 4 Уолтер знает, что в r не было включено никакой скрытой информации, но не сможет доказать этот факт третьей стороне, воспроизведя запись протокола. Более того, Уолтер, если захочет, может использовать этот протокол для создания собственного скрытого канала. Он может включить скрытую информацию в одну из подписей Алисы, выбрав k" с определенными характеристиками. Когда Симмонс открыл такую возможность, он назвал её «Каналом кукушки». Предотвратить «Канал кукушки» можно с помощью трехпроходного протокола генерации k.

**3 Тестирование программы**

На рисунках 1-2 представлены результаты работы программы, эмулирующей передачу секретного бита через секретный канал в протоколе DSA.

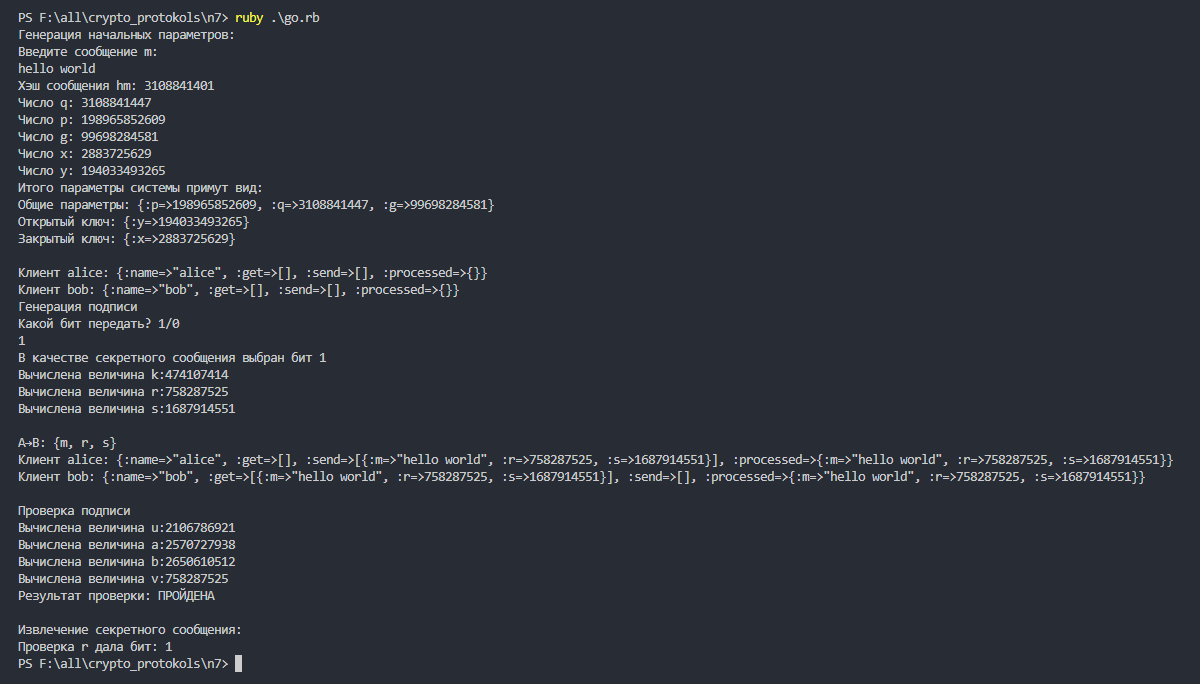


Рисунок 1 - Результаты работы программы

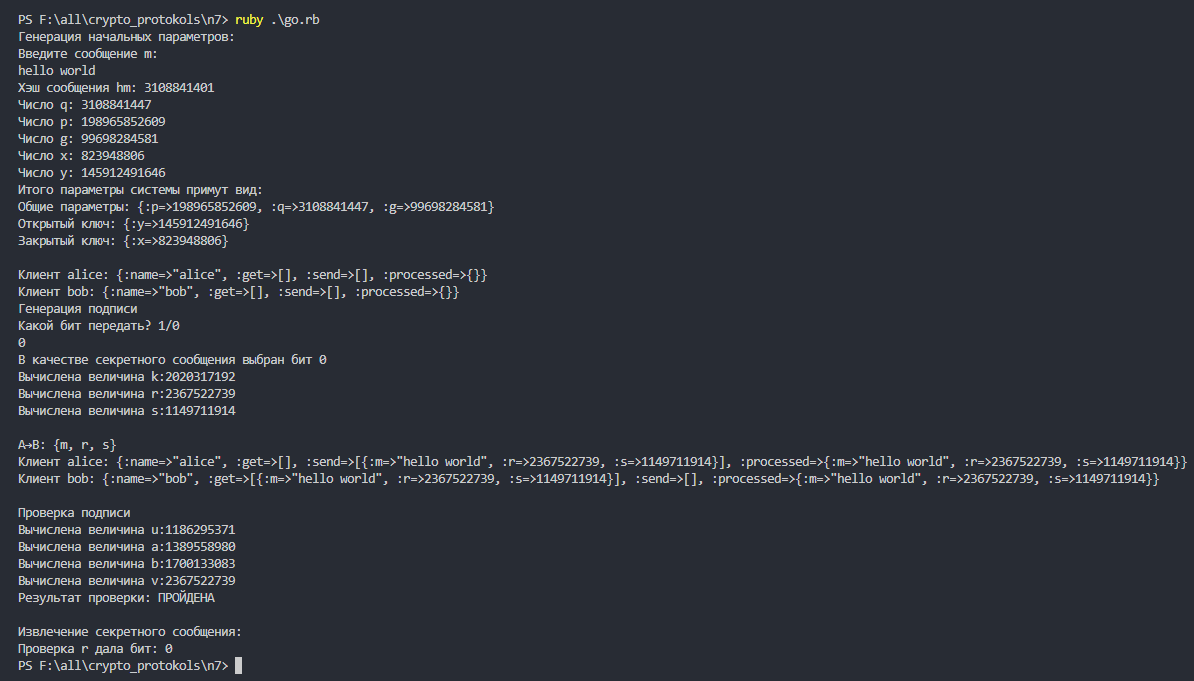


Рисунок 2 - Результаты работы программы

**ПРИЛОЖЕНИЕ А**

**Код программы steps.rb**

require 'prime'

require 'openssl'

class Steps

  def initialize(params = {})

    @bit\_length = params.dig(:bit\_length)

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

  end

  def step0

    puts "Генерация начальных параметров:"

    puts "Введите сообщение m:"

    @m = gets.strip.to\_s

    @hm = one\_way\_hash(@m, 8).to\_i(16)

    puts "Хэш сообщения hm: #{@hm}"

    q = get\_more\_prime(@hm)

*# q = next\_prime\_after(@hm)*

    puts "Число q: #{q}"

    p = gen\_p(q)

    puts "Число p: #{p}"

    g = 2.pow((p-1)/q) % p

    puts "Число g: #{g}"

    x = rand(q)

    puts "Число x: #{x}"

    y = g.pow(x, p)

    puts "Число y: #{y}"

    @main = {

      p: p,

      q: q,

      g: g

    }

    @secret\_key = {

      x: x

    }

    @open\_key = {

      y: y

    }

    puts "Итого параметры системы примут вид:"

    puts "Общие параметры: #{@main}"

    puts "Открытый ключ: #{@open\_key}"

    puts "Закрытый ключ: #{@secret\_key}"

    @alice = {name: "alice", get: [], send:[], processed: {}}

    @bob = {name: "bob", get: [], send:[], processed: {}}

    puts "\nКлиент #{@alice[:name]}: #{@alice}" if @debug\_mode

    puts "Клиент #{@bob[:name]}: #{@bob}\n" if @debug\_mode

  end

  def step1

    puts "Генерация подписи"

    puts "Какой бит передать? 1/0"

    bit = gets.strip.to\_i

    s = 0

    r = 0

    while s == 0 || r == 0

      k = rand(@main[:q])

      r = (@main[:g].pow(k, @main[:p])).pow(1, @main[:q])

      case bit

      when 1

        while lezhandr2(r, @main[:p]) != 1

          k = rand(@main[:q])

          r = (@main[:g].pow(k, @main[:p])).pow(1, @main[:q])

        end

      when 0

        while lezhandr2(r, @main[:p]) != -1

          k = rand(@main[:q])

          r = (@main[:g].pow(k, @main[:p])).pow(1, @main[:q])

        end

      end

      s = (inverse(k, @main[:q]) \* (@hm + @secret\_key[:x] \* r)) % @main[:q]

    end

    puts "В качестве секретного сообщения выбран бит #{bit}"

    puts "Вычислена величина k:#{k}"

    puts "Вычислена величина r:#{r}"

    puts "Вычислена величина s:#{s}"

    puts "\nA→B: {m, r, s}"

    @alice[:send] << {m: @m, r: r, s:s}

    @bob[:get] << @alice[:send].last

    @bob[:processed].merge!(@bob[:get].last)

    @alice[:processed].merge!(@alice[:send].last)

    @alice[:processed].merge!({m: @m, r: r, s:s})

    puts "Клиент #{@alice[:name]}: #{@alice}" if @debug\_mode

    puts "Клиент #{@bob[:name]}: #{@bob}\n" if @debug\_mode

  end

  def step2

    puts "\nПроверка подписи"

    data = @bob[:processed]

    u = inverse(data[:s], @main[:q])

    a = (@hm \* u) % @main[:q]

    b = (data[:r] \* u) % @main[:q]

    v = (@main[:g].pow(a, @main[:p]) \* @open\_key[:y].pow(b, @main[:p]) % @main[:p]) % @main[:q]

    puts "Вычислена величина u:#{u}"

    puts "Вычислена величина a:#{a}"

    puts "Вычислена величина b:#{b}"

    puts "Вычислена величина v:#{v}"

    puts "Результат проверки: #{v == data[:r] ? "ПРОЙДЕНА" : "НЕ ПРОЙДЕНА"}"

    puts "\nИзвлечение секретного сообщения:"

    puts "Проверка r дала бит: #{lezhandr2(data[:r], @main[:p]) == 1 ? "1" : "0"}"

  end

  private

  def gen\_random\_prime

    raise 'Not enough number length!' if @bit\_length == 0

    num = 1

    while !num.prime?

      bin\_str = '1'

      (@bit\_length - 1).times do

        bin\_str += rand(2).to\_s

      end

      num = bin\_str.to\_i(2)

    end

    num

  end

  def get\_more\_prime(n)

    if n % 2 == 0

      num = n + 1

    else

      num = n + 2

    end

    while !num.prime?

      num +=2

    end

    num

  end

  def one\_way\_hash(data, hash\_length = 64)

    raise ArgumentError, 'Некорректная длина хэша' unless hash\_length.positive?

    sha256 = OpenSSL::Digest::SHA256.new

    hashed\_data = sha256.digest(data)

*# Получаем хэш длиной hash\_length и возвращаем его*

    hashed\_data.unpack('H\*')[0][0, hash\_length]

  end

  def gen\_p(q)

*# Проверяем, что q является простым числом*

    raise ArgumentError, 'Входное число q не является простым' unless q.prime?

*# Генерируем простые числа, начиная с q+1, и проверяем, чтобы p-1 было делителем q*

*# p\_candidate = q + 1*

*# loop do*

*#   if p\_candidate.prime? && (p\_candidate - 1) % q == 0*

*#     return p\_candidate*

*#   else*

*#     p\_candidate += 1*

*#   end*

*# end*

    two = 2

    while !(two \* q + 1).prime?

      two \*= 2

    end

    two \* q + 1

  end

  def lezhandr2(a, p)

    return 0 if a % p == 0

    a.pow((p-1)/2, p) == 1 ? 1 : -1

  end

  def gcd\_ext(a, b, first = true)

    if a == 0

      return b, 0, 1

    else

      res, x, y = gcd\_ext(b%a, a, false)

      return res, y - (b / a) \* x, x

    end

  end

  def inverse(a, md)

    gcd, x, \_ = gcd\_ext(a, md)

    if gcd != 1

      raise "\nNo inverse element exists\n"

    else

      return x % md

    end

  end

  def next\_prime\_after(n)

    raise ArgumentError, 'Введите положительное число' unless n.is\_a?(Integer) && n.positive?

    primes = Prime.each.lazy

    next\_prime = nil

    primes.each do |prime|

      if prime > n

        next\_prime = prime

        break

      end

    end

    next\_prime

  end

end

**ПРИЛОЖЕНИЕ Б**

**Код программы go.rb**

require './steps.rb'

@steps = Steps.new(

  {

    debug\_mode:true

  }

)

@steps.step0

@steps.step1

@steps.step2