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

**Протоколы** **передачи секретного ключа по открытому каналу**

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 трехэтапного протокола Шамира передачи секретного ключа по открытому каналу и его программная реализация.

Задачи работы:

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

**2 Теоретические сведения**

Трёхэтапный протокол Шамира — криптографический трёхэтапный протокол, разработанный Ади Шамиром около 1980 года. Протокол позволяет двум сторонам безопасно обмениваться сообщениями без необходимости распространения ключей шифрования. Обмен сообщением между пользователями происходит в три прохода.

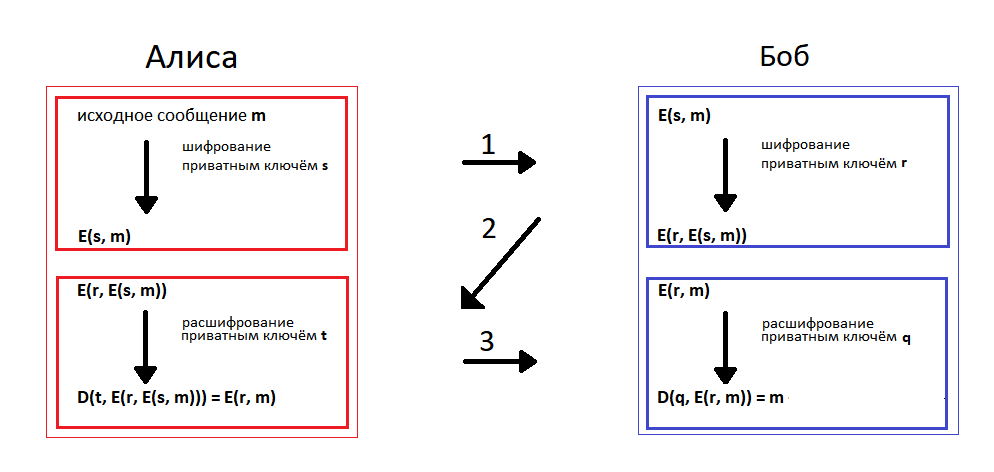


Рисунок 1 - Пример работы трехэтапного протокол Шамира

Используется шифрование на основе функции возведения в степень по модулю. Выбирают достаточно большое простое число для которого имеет большой простой множитель. В информационном взаимодействии участвуют два пользователя: Алиса и Боб.

1. Алиса выбирает число , взаимно простое с . Также Алиса использует число такое, что , то есть . Алиса шифрует сообщение и отправляет шифр Бобу:
2. Получатель Боб аналогично выбирает целое число , взаимно простое с , и число такое, что . Боб отправляет обратно следующее сообщение:
3. Алиса, получив сообщение, вычисляет (используется коммутативность функции возведения в степень по модулю и свойство по малой теореме Ферма) и отправляет Бобу:
4. Боб расшифровывает сообщение: .

Если третья сторона перехватила все три сообщения:

Чтобы вычислить при корректно выбранных параметрах и , нужно решить систему из этих трех уравнений, что имеет очень большую вычислительную сложность, так как нужно решать задачу дискретного логарифма.

**3 Тестирование программы**

На рисунках 2-3 представлены результаты работы программы, эмулирующей работу трехэтапного протокола Шамира между двумя субъектами обмена (Alice и Bob).



Рисунок 2 - Пример работы программы

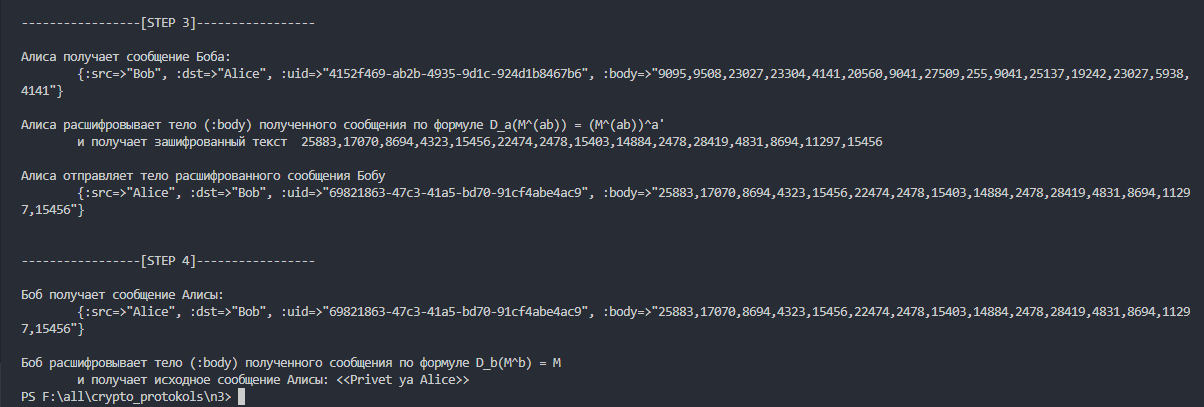


Рисунок 3 - Пример работы программы

**ПРИЛОЖЕНИЕ А**

**Код программы coder.rb**

class Coder

  def initialize(params = {})

    @p = params.dig(:p).to\_i

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

    @a, @b = generate\_ab(@p)

  end

  def encode(message, type = :raw)

    case type

    when :raw

      encoded\_message = message.bytes.map { |c| c.pow(@a, @p) }

      encoded\_message.join(",")

    when :enc

      encoded\_message = message.split(',').map { |c| c.to\_i.pow(@a, @p) }

      encoded\_message.join(",")

    end

  end

  def decode(encoded\_message, type = :raw)

    case type

    when :raw

      encoded\_message.split(",").map { |c| c.to\_i.pow(@b, @p) }.pack("C\*")

    when :enc

      decoded\_message = encoded\_message.split(",").map { |c| c.to\_i.pow(@b, @p) }

      decoded\_message.join(",")

    end

  end

  def get\_params

    {

      p: @p,

      a: @a,

      b: @b

    }

  end

  private

  def generate\_ab(p)

    a = find\_coprime(p - 1)

    b = modular\_inverse(a, p - 1)

    [a, b]

  end

  def find\_coprime(n)

    random = Random.new

    candidate = random.rand(2..n-1)

    until candidate.gcd(n) == 1

      candidate = (candidate + 1) % n

    end

    candidate

  end

  def modular\_inverse(a, p)

    x, y = extended\_gcd(a, p)

    x += p if x < 0

    x

  end

  def extended\_gcd(a, b)

    return [0, 1] if a % b == 0

    x, y = extended\_gcd(b, a % b)

    [y, x - (a / b) \* y]

  end

end

**ПРИЛОЖЕНИЕ Б**

**Код программы shamir.rb**

require './coder.rb'

require 'prime'

require 'securerandom'

require 'openssl'

class Shamir

  def initialize(params = {})

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

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

    @message = params.dig(:message)

    @int\_gen\_type = params.dig(:int\_gen\_type)

  end

  def start

  end

  def step0

    puts "\n\n-----------------[STEP 0]-----------------\n\n"

    prime = gen\_large\_p(@int\_gen\_type)

    @alice = make\_client('Alice', prime)

    @bob = make\_client('Bob', prime)

    puts "\nАлиса и Боб договариваются о большом простом числе P, таком, что:"

    puts "\t! P-1 имеет большой простой множитель\n\tP = #{prime}"

    puts "\nАлиса и Боб независимо выбирают числа a и b такие, что:"

    puts "\t! НОД(a, p - 1) == 1\n\t! a \* b == 1 (mod p - 1)"

    puts "\tАлиса: #{@alice[:coder].get\_params}\n\tБоб: #{@bob[:coder].get\_params}"

    if @debug\_mode == :all

      puts "\nСостояние Алисы"

      pp @alice

      puts "\nСостояние Боба"

      pp @bob

    end

    gets if @debug\_mode == :all || @debug\_mode == :by\_step

  end

  def step1

    puts "\n\n-----------------[STEP 1]-----------------\n\n"

    send\_message(@alice, @bob, @alice[:coder].encode(@message))

    puts "Алиса шифрует исходное сообщение <<#{@message}>> по формуле E\_a(M) = M^a (mod p)"

    puts "\nАлиса отправляет сообщение Бобу:\n\t#{@alice[:m\_pushed][-1]}"

    if @debug\_mode == :all

      puts "\nСостояние Алисы"

      pp @alice

      puts "\nСостояние Боба"

      pp @bob

    end

    gets if @debug\_mode == :all || @debug\_mode == :by\_step

  end

  def step2

    puts "\n\n-----------------[STEP 2]-----------------\n\n"

    last\_msg = @bob[:m\_pulled][-1][:body]

*# puts last\_msg*

    send\_message(@bob, @alice, @bob[:coder].encode(last\_msg, :enc))

    puts "Боб получает сообщение Алисы:\n\t#{@bob[:m\_pulled][-1]}"

    puts "\nБоб шифрует полученное от Алисы тело (:body) сообщения по формуле E\_b(E\_a(M)) = M^(ab) (mod p)"

    puts "\nБоб отправляет сообщение Алисе\n\t#{@bob[:m\_pushed][-1]}"

    if @debug\_mode == :all

      puts "\nСостояние Алисы"

      pp @alice

      puts "\nСостояние Боба"

      pp @bob

    end

    gets if @debug\_mode == :all || @debug\_mode == :by\_step

  end

  def step3

    puts "\n\n-----------------[STEP 3]-----------------\n\n"

    @alice[:m\_decoded] << process\_mesage(@alice, :enc)

    send\_message(@alice, @bob, @alice[:m\_decoded][-1][:body])

    puts "Алиса получает сообщение Боба:\n\t#{@alice[:m\_pulled][-1]}"

    puts "\nАлиса расшифровывает тело (:body) полученного сообщения по формуле D\_a(M^(ab)) = (M^(ab))^a'"

    puts "\tи получает зашифрованный текст\t#{@alice[:m\_decoded][-1][:body]}"

    puts "\nАлиса отправляет тело расшифрованного сообщения Бобу\n\t#{@alice[:m\_pushed][-1]}"

    if @debug\_mode == :all

      puts "\nСостояние Алисы"

      pp @alice

      puts "\nСостояние Боба"

      pp @bob

    end

    gets if @debug\_mode == :all || @debug\_mode == :by\_step

  end

  def step4

    puts "\n\n-----------------[STEP 4]-----------------\n\n"

    @bob[:m\_decoded] << process\_mesage(@bob, :raw)

    puts "Боб получает сообщение Алисы:\n\t#{@bob[:m\_pulled][-1]}"

    puts "\nБоб расшифровывает тело (:body) полученного сообщения по формуле D\_b(M^b) = M"

    puts "\tи получает исходное сообщение Алисы: <<#{@bob[:m\_decoded][-1][:body]}>>"

    if @debug\_mode == :all

      puts "\nСостояние Алисы"

      pp @alice

      puts "\nСостояние Боба"

      pp @bob

    end

    gets if @debug\_mode == :all || @debug\_mode == :by\_step

    if @debug\_mode == :by\_step

      puts "\nСостояние Алисы"

      pp @alice

      puts "\nСостояние Боба"

      pp @bob

    end

  end

  private

  def make\_client(name = '', p)

    {

      p: p,

      name: name,

      coder: Coder.new({p: p, bit\_length: @bit\_length}),

      m\_pushed: [],

      m\_pulled: [],

      m\_decoded: []

    }

  end

  def send\_message(src, dst, m)

    message = {

      src: src[:name],

      dst: dst[:name],

      uid: SecureRandom.uuid,

      body: m

    }

    src[:m\_pushed] << message

    dst[:m\_pulled] << message

  end

  def process\_mesage(client, type = :raw)

    last\_message = client[:m\_pulled][-1]

    {

      src: last\_message[:src],

      dst: last\_message[:dst],

      uid: last\_message[:uid],

      body: client[:coder].decode(last\_message[:body], type)

    }

  end

  def gen\_large\_p(mode = :default)

    case mode

    when :default

      loop do

        candidate = rand(2 \*\* @bit\_length)

*# pp candidate*

        next if candidate.even?

        if Prime.prime?(candidate)

          factors = Prime.prime\_division(candidate - 1)

          largest\_prime\_factor = factors[-1][0]

          return candidate if largest\_prime\_factor > Math.sqrt(candidate).to\_i

        end

      end

    when :alt

      OpenSSL::BN.generate\_prime(@bit\_length).to\_i

    end

  end

end

**ПРИЛОЖЕНИЕ В**

**Код программы process.rb**

require './shamir.rb'

puts "\nEnter bin length"

lng = gets.strip.to\_i

puts "\nEnter Alice's message:"

message = gets.strip.to\_s

puts "\nEnter debug\_mode type"

debug\_mode = gets.strip.to\_s

params = {

  bit\_length: lng,

  debug\_mode: debug\_mode != '' ? debug\_mode : 'by\_step',

  message: message,

  int\_gen\_type: :alt

}

shamir = Shamir.new(params)

shamir.step0

shamir.step1

shamir.step2

shamir.step3

shamir.step4