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 ЭЦП**

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 схемы подписи Гиллу-Кискате и ее программная реализация.

Задачи работы:

* Изучить схему подписи Гиллу-Кискате, ее сильные и слабые стороны;
* Привести программную реализацию схемы.

**2 Теоретические сведения**

Протокол Guillou-Quisquater — это протокол идентификации с нулевым разглашением, расширение более раннего протокола Фиата — Шамира, разработанный Луи Гиллу (англ. Louis Guillou), Жан-Жак Кискатр (англ. Jean-Jacques Quisquater) в 1988 году.

Протокол позволяет одному участнику (доказывающему A) доказать другому участнику (проверяющему B), что он обладает секретной информацией, не раскрывая ни единого бита этой информации.

Безопасность протокола основана на сложности извлечения квадратного корня по модулю достаточно большого составного числа по заданному модулю n.

В сравнении с протоколом Фиата-Шамира протокол Guillou-Quisquater имеет меньшее число сообщений, которыми необходимо поменяться сторонам для идентификации. Протокол требует только один раунд обмена сообщениями, имеет более низкие требования к памяти, используемой для хранения секретов пользователей, однако требует большего объёма вычислений.

Кроме того, схему идентификации Guillou-Quisquater можно легко преобразовать в схему подписи. Далее будем рассматривать только многократную схему подписи Гиллу-Кискате.

Алиса (A) и Боб (B) подписывают сообщение , с тем, чтобы в дальнейшем Кэрол (C) могла проверить их совместную подпись.

**Генерация общих параметров.**

Доверенный центр (Трент) выбирает большое число , где , — большие различные простые числа, которые держатся в секрете. выбирает целое число , взаимно простое с , где — функция Эйлера. Параметры объявляются открытыми и общими для подписантов.

**Генерация индивидуальных параметров.**

вычисляет . Затем вычисляет закрытый ключ Алисы , где — открытый ключ Алисы (битовая строка личной информации о пользователе с условием ), и закрытый ключ Боба , где — открытый ключ Боба (битовая строка личной информации о пользователе с условием ). Индивидуальными параметрами соответственно являются и .

**Генерация подписи.**

1. : , где и — случайное число Алисы, ;

2. : , где и — случайное число Боба, ;

3. : , (Алиса и Боб вычисляют вычисляют );

4. : , ;

5. : , где ;

6. : , где ;

7. : {z}, где ;

8. : .

**Проверка подписи.**

9. : , где ;

10. : , где ;

11. : , где ;

12. : проверяет, что .

**3 Тестирование программы**

На рисунках 1-3 представлены результаты работы программы, эмулирующей работу схемы подписи Гиллу-Кискате.

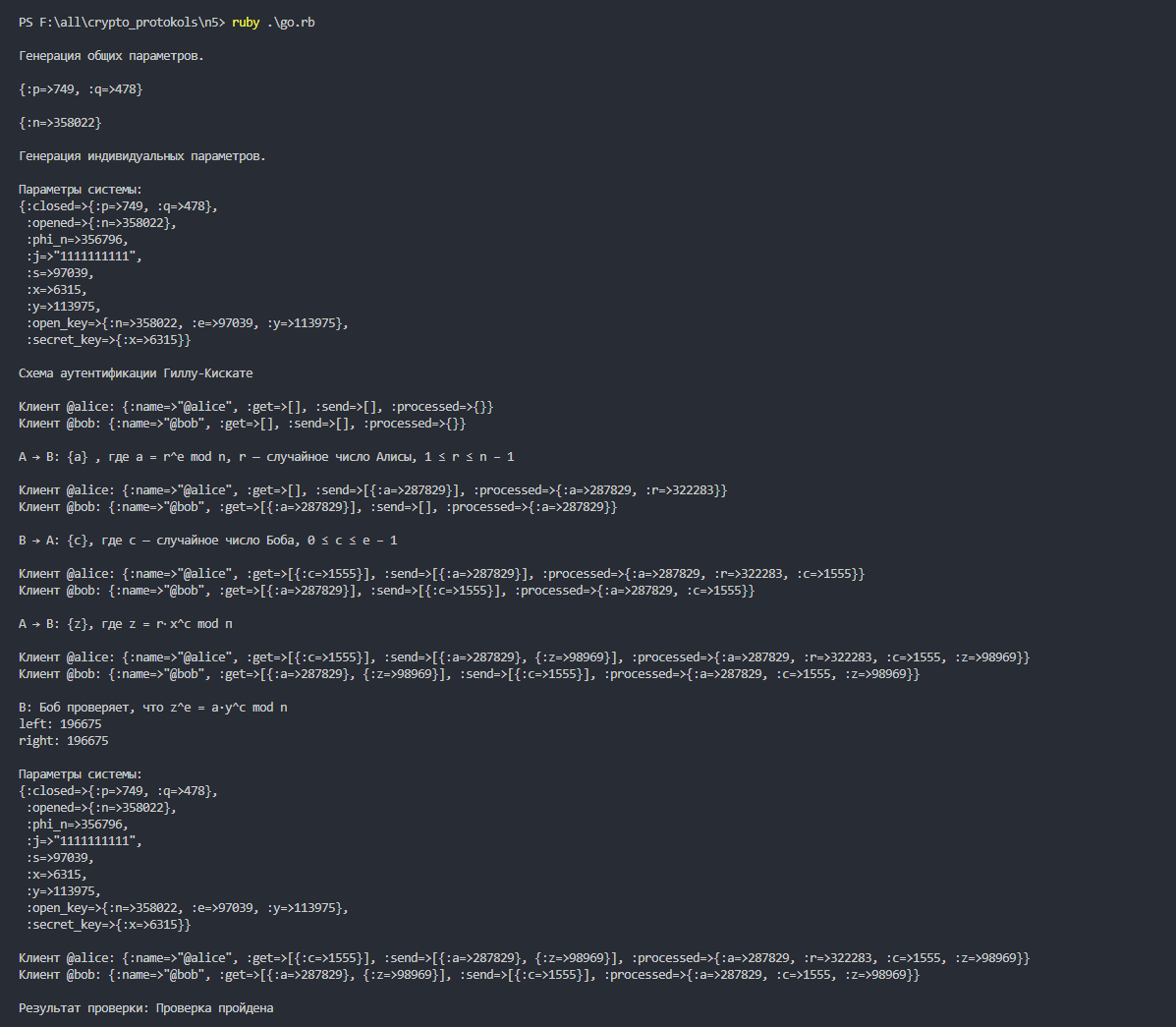


Рисунок 1 - Пример работы программы

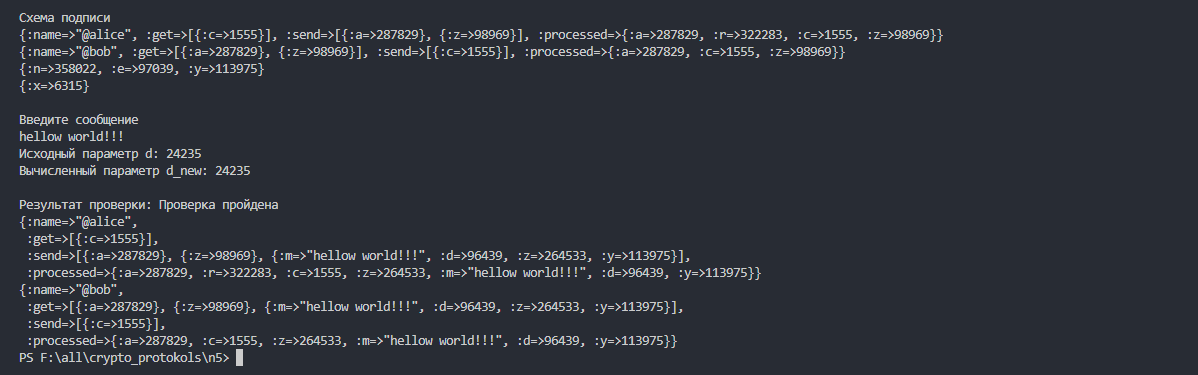


Рисунок 2 - Пример работы программы

**ПРИЛОЖЕНИЕ А**

**Код программы go.rb**

require './steps.rb'

steps = Steps.new(

  {

    debug\_mode: true,

    bit\_length: 10

  }

)

steps.step0

steps.step1

steps.step2

steps.step3

**ПРИЛОЖЕНИЕ Б**

**Код программы steps.rb**

require 'prime'

require 'openssl'

require 'securerandom'

class Steps

  def initialize(params = {})

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

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

  end

  def step0

    puts "\nГенерация общих параметров."

    gen\_close\_params

    puts "\n#{@close\_params}"

    gen\_open\_params

    puts "\n#{@open\_params}"

  end

  def step1

    puts "\nГенерация индивидуальных параметров."

    phi\_n = ez\_mult(@close\_params[:p] - 1, @close\_params[:q] - 1)

    e = generate\_coprime\_number(phi\_n)

    big\_j = generate\_coprime\_bitstring(@open\_params[:n])

    s = mod\_pow\_inverse(e, 1, phi\_n)

    x = mod\_pow\_inverse(big\_j.to\_i(2), s, @open\_params[:n])

    y = x.pow(e, @open\_params[:n])

    @all\_params = {

      closed: @close\_params,

      opened: @open\_params,

      phi\_n: phi\_n,

      j: big\_j,

      s: s,

      x: x,

      y: y

    }

    @open\_key = {

      n: @open\_params[:n],

      e: e,

      y: y

    }

    @secret\_key = {

      x: x

    }

    @all\_params.merge!(

      {

        open\_key: @open\_key,

        secret\_key: @secret\_key

      }

    )

    puts "\nПараметры системы:"

    pp @all\_params

  end

  def step2

    puts "\nСхема аутентификации Гиллу-Кискате"

    @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

    puts "\nA → B: {a} , где a = r^e mod n, r — случайное число Алисы, 1 ≤ r ≤ n – 1"

    r = rand(1..(@open\_key[:n]))

    a = r.pow(@open\_key[:e], @open\_key[:n])

    @alice[:send] << {a: a}

    @bob[:get] << @alice[:send].last

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

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

    @alice[:processed].merge!({r: r})

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

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

    puts "\nB → A: {c}, где c — случайное число Боба, 0 ≤ c ≤ e – 1"

    c = rand(0..(@open\_key[:e]-1))

    @bob[:send] << {c: c}

    @alice[:get] << @bob[:send].last

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

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

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

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

    puts "\nA → B: {z}, где z = r⋅x^c mod п"

    z = r \* @secret\_key[:x].pow(@alice[:processed][:c]) % @open\_key[:n]

    @alice[:send] << {z: z}

    @bob[:get] << @alice[:send].last

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

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

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

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

    puts "\nB: Боб проверяет, что z^e = a·у^c mod n"

    puts "left: #{@bob[:processed][:z].pow(@open\_key[:e]) % @open\_key[:n]}" if @debug\_mode

    puts "right: #{(@bob[:processed][:a]\*@open\_key[:y].pow(@bob[:processed][:c]) % @open\_key[:n])}" if @debug\_mode

    checkout = (

      (@bob[:processed][:z].pow(@open\_key[:e]) % @open\_key[:n]) ==

      (@bob[:processed][:a]\*@open\_key[:y].pow(@bob[:processed][:c]) % @open\_key[:n])

    )

    puts "\nПараметры системы:" if @debug\_mode

    pp @all\_params if @debug\_mode

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

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

    puts "\nРезультат проверки: " + (checkout ? "Проверка пройдена" : "Проверка не пройдена")

  end

  def step3

    puts "\nСхема подписи"

    pp @alice

    pp @bob

    pp @open\_key

    pp @secret\_key

    puts "\nВведите сообщение"

    message = gets.strip.to\_s

    a = @alice[:processed][:r].pow(@open\_key[:e], @open\_key[:n])

    d = one\_way\_hash(message + a.to\_s) % @open\_key[:e]

    z = (@alice[:processed][:r] \* @secret\_key[:x]).pow(d, @open\_key[:n])

    @alice[:send] << {m: message, d: d, z: z, y: @open\_key[:y]}

    @bob[:get] << @alice[:send].last

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

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

    a\_new = z.pow(@open\_key[:e], @open\_key[:n]) \* @open\_key[:y].pow(d, @open\_key[:n])

    d\_new = one\_way\_hash(message + a\_new.to\_s) % @open\_key[:e]

    checkout = d = d\_new

    puts "Исходный параметр d: #{d}"

    puts "Вычисленный параметр d\_new: #{d\_new}"

    puts "\nРезультат проверки: " + (checkout ? "Проверка пройдена" : "Проверка не пройдена")

    pp @alice

    pp @bob

  end

  private

  def one\_way\_hash(data)

    sha256 = OpenSSL::Digest::SHA256.new

    hashed\_data = sha256.digest(data)

    hashed\_data.unpack('H\*')[0].to\_i(16)

  end

  def mod\_pow\_inverse(x, a, n)

    raise ArgumentError, "n should be greater than 1" if n <= 1

    raise ArgumentError, "a should be a non-negative integer" if a < 0

    result = 1

    base = x % n

    while a > 0

      result = (result \* base) % n if a.odd?

      base = (base \* base) % n

      a /= 2

    end

    result

  end

  def mod\_inverse(a, m)

    m0, x0, x1 = m, 0, 1

    while a > 1

      q = a / m

      m, a = a % m, m

      x0, x1 = x1 - q \* x0, x0

    end

    x1 += m0 if x1 < 0

    x1

  end

  def generate\_coprime\_number(n)

    raise ArgumentError, "n should be greater than 1" if n <= 1

*# Генерируем случайное число*

    random\_number = rand(2..n-1)

*# Проверяем взаимную простоту с n*

    until random\_number.gcd(n) == 1

      random\_number = rand(2..n-1)

    end

    return random\_number

  end

  def generate\_coprime\_bitstring(n)

    raise ArgumentError, "n should be greater than 1" if n <= 1

*# Генерируем случайную битовую строку*

    random\_bitstring = rand(2\*\*@bit\_length).to\_s(2)

*# Проверяем взаимную простоту с n*

    until random\_bitstring.to\_i.gcd(n) == 1

      random\_bitstring = rand(2\*\*@bit\_length).to\_s(2)

    end

    return random\_bitstring

  end

  def gen\_close\_params

    @close\_params = {

      p: gen\_big\_num(@bit\_length),

      q: gen\_big\_num(@bit\_length)

    }

  end

  def gen\_open\_params

    @open\_params = {

      n: ez\_mult(@close\_params[:p], @close\_params[:q])

    }

  end

  def gen\_big\_num(bit\_length = 15)

    raise ArgumentError, "Bit length should be greater than 0" if bit\_length <= 0

*# Генерируем случайное число с использованием SecureRandom*

    random\_number = SecureRandom.random\_number(2\*\*bit\_length)

    return random\_number

  end

  def ez\_mult(x, y)

*# Базовый случай: если числа состоят из одной цифры*

    return x \* y if x < 10 || y < 10

*# Находим количество цифр в числах*

    m = [x.to\_s.length, y.to\_s.length].max

    m2 = (m / 2).to\_i

*# Разбиваем числа на две части*

    high1, low1 = x.divmod(10\*\*m2)

    high2, low2 = y.divmod(10\*\*m2)

*# Рекурсивно вычисляем три произведения*

    z0 = ez\_mult(low1, low2)

    z1 = ez\_mult((low1 + high1), (low2 + high2))

    z2 = ez\_mult(high1, high2)

*# Применяем формулу Карацубы для вычисления конечного результата*

    return (z2 \* 10\*\*(2 \* m2)) + ((z1 - z2 - z0) \* 10\*\*m2) + z0

  end

  def verify\_signature(message, signature)

    a = @open\_key[:n].pow(@open\_key[:e], @open\_key[:n])

    z = @secret\_key[:x].pow(Integer(signature, 16), @open\_key[:n])

    expected\_hash = one\_way\_hash(message + a.to\_s + @all\_params[:s].to\_s)

    return z == expected\_hash.to\_i(16)

  end

end