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 m, с тем, чтобы в дальнейшем Кэрол (C) могла проверить их совместную подпись.

Генерация общих параметров.

Доверенный центр T (Трент) выбирает большое число $n = p \cdot q$, где p, q — большие различные простые числа, которые держатся в секрете. T выбирает целое число e $(1 < e < \varphi(n))$, взаимно простое с $\varphi(n)$, где $\varphi(n) = (p-1)(q-1)$ — функция Эйлера. Параметры $\{n, e\}$ объявляются открытыми и общими для подписантов.

Генерация индивидуальных параметров.

Т вычисляет $s = e^{(-1)}\big(mod\ \varphi(n)\big)$. Затем Т вычисляет закрытый ключ Алисы $x_A = J_A^{-s}\ (mod\ n)$, где J_A — открытый ключ Алисы (битовая строка личной информации о пользователе A с условием $(J_A, n) = 1$), и закрытый ключ Боба $x_B = J_B^{-s}(mod\ n)$, где J_B — открытый ключ Боба (битовая строка личной информации о пользователе B с условием $(J_B, n) = 1$). Индивидуальными параметрами соответственно являются $\{J_A, x_A\}$ и $\{J_B, x_B\}$.

Генерация подписи.

1. $A \to B$: $\{a_A\}$, где $a_A = r_A^e\ mod\ n$ и r_A — случайное число Алисы, $1 \leq r_A \leq n-1$;

2. $B \to A$: $\{a_B\}$, где $a_B = r_B^e\ mod\ n$ и r_B — случайное число Боба, $1 \leq r_B \leq n-1$;

3. A, B: $\{a\}$, $a = a_A \cdot a_B\ mod\ n$ (Алиса и Боб вычисляют вычисляют a);

4. A, B: $\{d\}$, $d = h\ (m||a)\ mod\ e$;

5. $A \to B$: $\{z_A\}$, где $z_A = r_A \cdot x_A^d\ mod\ n$;

6. $B \to A$: $\{z_B\}$, где $z_B = r_B \cdot x_B^d\ mod\ n$;

7. A, B: {z}, где $z = z_A \cdot z_B\ mod\ n$;

8. $A, B \to C$: $\{m, d, z, J_A, J_B\}$.

Проверка подписи.

9. C: $\{J\}$, где $J = J_A \cdot J_B\ mod\ n$;

10. C: $\{a^*\}$, где $a^* = z^e \cdot J^d\ mod\ n$;

11. C: $\{d^*\}$, где $d^* = h(m||a^*)\ mod\ e$;

12. C: проверяет, что $\mathrm{d}^* = d$.

3 Тестирование программы

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

```
PS F:\all\crypto_protokols\n5> ruby .\go.rb

Генерация общих параметров.

{:p=>749, :q=>478}

{:n=>358022}

Генерация индивидуальных параметров.

Параметры системы:
{:closed=>{:p=>749, :q=>478},
 :opened=>{:n=>358022},
 :phi_n=>356796,
 :j=>"1111111111",
 :s=>97039,
 :x=>6315,
 :y=>113975,
 :open_key=>{:n=>358022, :e=>97039, :y=>113975},
 :secret_key=>{:x=>6315}}

Схема аутентификации Гиллу-Кискате

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

A → B: {a} , где a = r^e mod n, r — случайное число Алисы, 1 ≤ r ≤ n − 1

Клиент @alice: {:name=>"@alice", :get=>[], :send=>[{:a=>287829}], :processed=>{:a=>287829, :r=>322283}}
Клиент @bob: {:name=>"@bob", :get=>[{:a=>287829}], :send=>[], :processed=>{:a=>287829}}

B → A: {c}, где c — случайное число Боба, 0 ≤ c ≤ e − 1

Клиент @alice: {:name=>"@alice", :get=>[{:c=>1555}], :send=>[{:a=>287829}], :processed=>{:a=>287829, :r=>322283, :c=>1555}}
Клиент @bob: {:name=>"@bob", :get=>[{:a=>287829}], :send=>[{:c=>1555}], :processed=>{:a=>287829, :c=>1555}}

A → B: {z}, где z = r·x^c mod n

Клиент @alice: {:name=>"@alice", :get=>[{:c=>1555}], :send=>[{:a=>287829}, {:z=>98969}], :processed=>{:a=>287829, :r=>322283, :c=>1555, :z=>98969}}
Клиент @bob: {:name=>"@bob", :get=>[{:a=>287829}, {:z=>98969}], :send=>[{:c=>1555}], :processed=>{:a=>287829, :c=>1555, :z=>98969}}

B: Боб проверяет, что z^e = a·y^c mod n
left: 196675
right: 196675

Параметры системы:
{:closed=>{:p=>749, :q=>478},
 :opened=>{:n=>358022},
 :phi_n=>356796,
 :j=>"1111111111",
 :s=>97039,
 :x=>6315,
 :y=>113975,
 :open_key=>{:n=>358022, :e=>97039, :y=>113975},
 :secret_key=>{:x=>6315}}

Клиент @alice: {:name=>"@alice", :get=>[{:c=>1555}], :send=>[{:a=>287829}, {:z=>98969}], :processed=>{:a=>287829, :r=>322283, :c=>1555, :z=>98969}}
Клиент @bob: {:name=>"@bob", :get=>[{:a=>287829}, {:z=>98969}], :send=>[{:c=>1555}], :processed=>{:a=>287829, :c=>1555, :z=>98969}}

Результат проверки: Проверка пройдена
```

Рисунок 1 - Пример работы программы

```
Схема подписи
{:name=>"@alice", :get=>[{:c=>1555}], :send=>[{:a=>287829}, {:z=>98969}], :processed=>{:a=>287829, :r=>322283, :c=>1555, :z=>98969}}
{:name=>"@bob", :get=>[{:a=>287829}, {:z=>98969}], :send=>[{:c=>1555}], :processed=>{:a=>287829, :c=>1555, :z=>98969}}
{:n=>358022, :e=>97039, :y=>113975}
{:x=>6315}

Введите сообщение
hellow world!!!
Исходный параметр d: 24235
Вычисленный параметр d_new: 24235

Результат проверки: Проверка пройдена
{:name=>"@alice",
 :get=>[{:c=>1555}],
 :send=>[{:a=>287829}, {:z=>98969}, {:m=>"hellow world!!!", :d=>96439, :z=>264533, :y=>113975}],
 :processed=>{:a=>287829, :r=>322283, :c=>1555, :z=>264533, :m=>"hellow world!!!", :d=>96439, :y=>113975}}
{:name=>"@bob",
 :get=>[{:a=>287829}, {:z=>98969}, {:m=>"hellow world!!!", :d=>96439, :z=>264533, :y=>113975}],
 :send=>[{:c=>1555}],
 :processed=>{:a=>287829, :c=>1555, :z=>264533, :m=>"hellow world!!!", :d=>96439, :y=>113975}}
PS F:\all\crypto_protokols\n5>
```

Рисунок 2 - Пример работы программы

ПРИЛОЖЕНИЕ А
Код программы go.rb

```ruby
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

```ruby
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 n"
```

```
    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·y^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]
```

```ruby
    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)
```

```ruby
  # 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
```

```ruby
    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
```