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 m, Боб проверяет подпись. Генерируются: q — большое простое число с условием $h = H(m) < q$; p — большое простое число , такого, что $(p-1)$ делится на q, а именно имеет вид $p - 1 = 2^t * q$. Выбирается число g такое, что его мультипликативный порядок по модулю p равен q. Для его вычисления можно воспользоваться формулой $g = h(p-1)/q \ mod \ p$, где h — некоторое произвольное число, $h \in (1; p-1)$ такое, что $g \neq 1$. В большинстве случаев значение $h = 2$ удовлетворяет этому требованию. Далее выбирается случайное число $x \in (0; q)$ и вычисляется $y = g\hat{}x \ mod \ p$. Элементы $\{p, q, g\}$ могут быть общими для группы пользователей, Элемент $\{y\}$ объявляется открытым ключом, элементы $\{x\}$ — закрытым ключом Алисы.

Генерация подписи.

1. A: $\{k\}$, выбирается случайное число $k \in (0;\ q)$;

2. A: {r}, где $r = (g\hat{}k\ mod\ p)\ mod\ q$;

3. A: $\{s\}$, где $s = k^{-1} * (H(m) + x * r)\ mod\ q$;

4. A: если $r = 0$ или $s = 0$, то выбор нового k;

5. $A \rightarrow B$: $\{m, r, s\}$.

Проверка подписи.

6. B : $\{u\}$, где $u = s - 1\ mod\ q$;

7. B : $\{a\}$, где $a = H(m) * u\ mod\ q$;

8. B : $\{b\}$, где $b = r * u\ mod\ q$;

9. B : $\{v\}$, где $v = (g^a * y^b\ mod\ p)\ mod\ q$;

10. B : если $v = r$, то подпись верна.

Скрытый канал на основе DSA

Основные параметры схемы в случае скрытого канала те же, только с добавлением простого числа P (отличающегося от параметра p 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 m и на шагах 1 и 2 выбирает случайное число $k \in (0; q)$ так, чтобы параметр r подписи являлся квадратичным вычетом по модулю P, если она хочет передать скрытый бит 1, или являлся являлся квадратичным невычетом по модулю P, 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 r квадратичным вычетом по модулю P и восстанавливает скрытый бит. Передача описанным выше способом нескольких битов

$b_1, b_2, \dots, b_n$ подразумевает подбор такого значения r, которое является квадратичным вычетом или невычетом по нескольким модулям $P_1, P_2, \dots, P_n$.

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. $A \rightarrow W$: $\{u\}$, где $u = g^{k'}$ и k' случайное число $k' \in (0;\ q)$;

2. $W \rightarrow A$: $\{k''\}$, где k'' случайное число $k'' \in (0;\ q)$;

3. A: $\{k\}$, где $k = k' \cdot k''\ mod$ (p − 1), далее продолжается основной протокол подписи DSA с шага 2, когда наступает время проверки подписи Уолтером, этот протокол продолжается;

4. W: $\{r'\}$, где $r' = (n^{k''}\ mod\ p)\ mod\ q$; если $r = r'$, 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.

```
PS F:\all\crypto_protokols\n7> ruby .\go.rb
Генерация начальных параметров:
Введите сообщение m:
hello world
Хэш сообщения hm: 3108841401
Число q: 3108841447
Число p: 198965852609
Число g: 99698284581
Число x: 2883725629
Число y: 194033493265
Итого параметры системы примут вид:
Общие параметры: {:p=>198965852609, :q=>3108841447, :g=>99698284581}
Открытый ключ: {:y=>194033493265}
Закрытый ключ: {:x=>2883725629}

Клиент alice: {:name=>"alice", :get=>[], :send=>[], :processed=>{}}
Клиент bob: {:name=>"bob", :get=>[], :send=>[], :processed=>{}}
Генерация подписи
Какой бит передать? 1/0
1
В качестве секретного сообщения выбран бит 1
Вычислена величина k:474107414
Вычислена величина r:758287525
Вычислена величина s:1687914551

A→B: {m, r, s}
Клиент alice: {:name=>"alice", :get=>[], :send=>[{:m=>"hello world", :r=>758287525, :s=>1687914551}], :processed=>{:m=>"hello world", :r=>758287525, :s=>1687914551}}
Клиент bob: {:name=>"bob", :get=>[{:m=>"hello world", :r=>758287525, :s=>1687914551}], :send=>[], :processed=>{:m=>"hello world", :r=>758287525, :s=>1687914551}}

Проверка подписи
Вычислена величина u:2106786921
Вычислена величина a:2570727938
Вычислена величина b:2650610512
Вычислена величина v:758287525
Результат проверки: ПРОЙДЕНА

Извлечение секретного сообщения:
Проверка r дала бит: 1
PS F:\all\crypto_protokols\n7>
```

Рисунок 1 - Результаты работы программы

```
PS F:\all\crypto_protokols\n7> ruby .\go.rb
Генерация начальных параметров:
Введите сообщение m:
hello world
Хэш сообщения hm: 3108841401
Число q: 3108841447
Число p: 198965852609
Число g: 99698284581
Число x: 823948806
Число y: 145912491646
Итого параметры системы примут вид:
Общие параметры: {:p=>198965852609, :q=>3108841447, :g=>99698284581}
Открытый ключ: {:y=>145912491646}
Закрытый ключ: {:x=>823948806}

Клиент alice: {:name=>"alice", :get=>[], :send=>[], :processed=>{}}
Клиент bob: {:name=>"bob", :get=>[], :send=>[], :processed=>{}}
Генерация подписи
Какой бит передать? 1/0
0
В качестве секретного сообщения выбран бит 0
Вычислена величина k:2020317192
Вычислена величина r:2367522739
Вычислена величина s:1149711914

A→B: {m, r, s}
Клиент alice: {:name=>"alice", :get=>[], :send=>[{:m=>"hello world", :r=>2367522739, :s=>1149711914}], :processed=>{:m=>"hello world", :r=>2367522739, :s=>1149711914}}
Клиент bob: {:name=>"bob", :get=>[{:m=>"hello world", :r=>2367522739, :s=>1149711914}], :send=>[], :processed=>{:m=>"hello world", :r=>2367522739, :s=>1149711914}}

Проверка подписи
Вычислена величина u:1186295371
Вычислена величина a:1389558980
Вычислена величина b:1700133083
Вычислена величина v:2367522739
Результат проверки: ПРОЙДЕНА

Извлечение секретного сообщения:
Проверка r дала бит: 0
PS F:\all\crypto_protokols\n7>
```

Рисунок 2 - Результаты работы программы

ПРИЛОЖЕНИЕ А
Код программы steps.rb

```ruby
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:"
```

```ruby
  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?
```

```ruby
      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

```ruby
require './steps.rb'

@steps = Steps.new(
  {
    debug_mode:true
  }
)

@steps.step0
@steps.step1
@steps.step2
```