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

**Разделение секрета**

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 разделения секрета Блэкли и ее программная реализация.

Задачи работы:

* Изучить схему разделения секрета Блэкли, ее сильные и слабые стороны;
* Привести программную реализацию схемы.

**2 Теоретические сведения**

Векторная схема разделения секрета, или схема Блэкли (англ. Blakley's scheme), — схема разделения секрета между сторонами, основанная на использовании точек многомерного пространства. Предложена Джорджем Блэкли в 1979 году. Схема Блэкли позволяет создать пороговое разделение секрета для любых , .

Разделяемым секретом в схеме Блэкли является одна из координат точки в -мерном пространстве. Долями секрета, раздаваемые сторонам, являются уравнения мерных гиперплоскостей. Для восстановления точки необходимо знать уравнений гиперплоскостей. Менее, чем сторон не смогут восстановить секрет, так как множеством пересечения плоскостей является прямая, и секрет не может быть восстановлен.

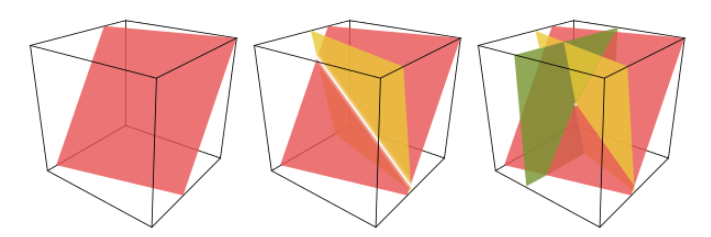


Рисунок 1 - Пример схемы Блэкли в трёх измерениях: каждая доля секрета — это плоскость, а секрет — это одна из координат точки пересечения плоскостей. Двух плоскостей недостаточно для определения точки пересечения.

Нужно отметить, что геометрическое описание и Рисунок 1 приведены для понимания главной идеи схемы. Однако сам процесс разделения секрета происходит в конечных полях с использованием аналогичного, но иного математического аппарата.

**Генерация точки**

Пусть нужно реализовать пороговую схему, то есть секрет разделить между сторонами так, чтобы любые из них могли восстановить секрет. Для этого выбирается большое простое число , по модулю которого будет строиться поле . Случайным образом дилер выбирает числа . Тем самым задается точка в мерном пространстве, первая координата которой является секретом.

**Раздача секрета**

Для каждой стороны случайным образом выбираются коэффициенты , равномерно распределённые в поле . Так как уравнение плоскости имеет вид , для каждой стороны необходимо вычислить коэффициенты :

При этом необходимо следить, чтобы любые уравнений были линейно независимы. В качестве долей секрета сторонам раздают набор коэффициентов, задающих уравнение гиперплоскости.

**Восстановление секрета**

Для восстановления секрета любым сторонам необходимо собраться вместе и из имеющихся долей секрета составить уравнения для отыскания точки пересечения гиперплоскостей:

Решение системы даёт точку в мерном пространстве, первая координата которой и есть разделяемый секрет. Систему можно решать любым известным способом, например, методом Гаусса, но при этом необходимо проводить вычисления в поле .

Если число участников встречи будет меньше, чем , например, , то результатом решения системы уравнений, составленной из имеющегося набора коэффициентов, будет прямая в мерном пространстве. Тем самым множество допустимых значений секрета, удовлетворяющих полученной системе, в точности совпадает с полным числом элементов поля , и секрет равновероятно может принимать любое значение из этого поля. Таким образом, участники, собравшись вместе, не получат никакой новой информации о разделённом секрете.

**3 Тестирование программы**

На рисунках 2-3 представлены результаты работы программы, эмулирующей работу схемы разделения секрета Блэкли.

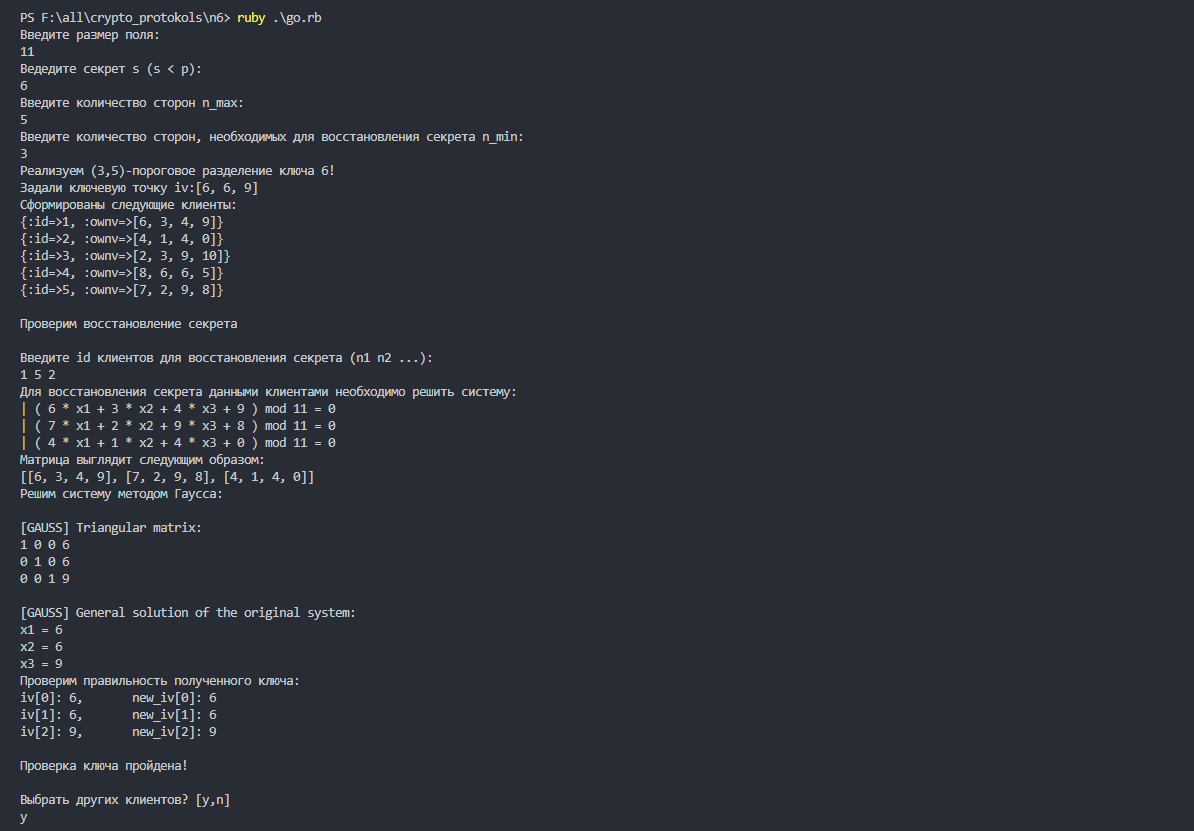


Рисунок 2 - Пример работы программы

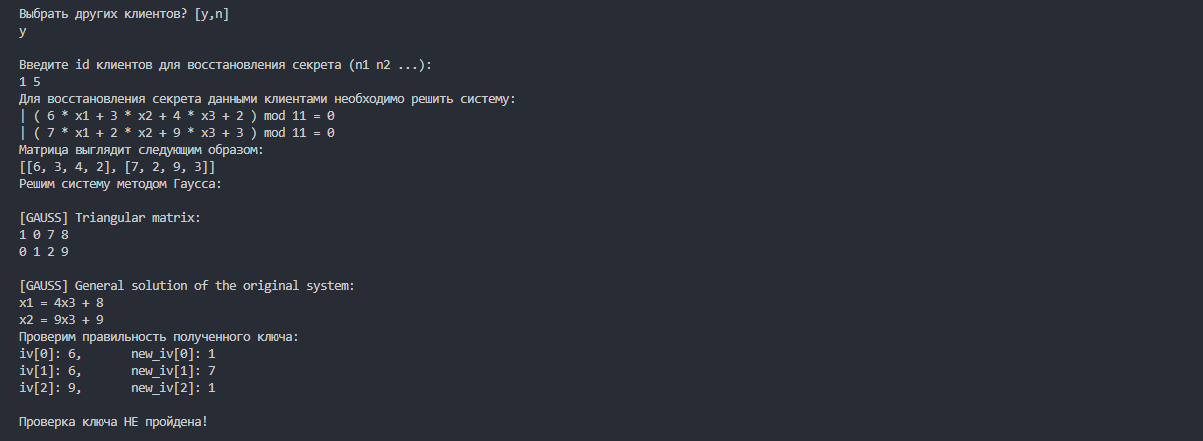


Рисунок 3 - Пример работы программы

**ПРИЛОЖЕНИЕ А**

**Код программы go.rb**

require './steps.rb'

@steps = Steps.new(

  {

    debug\_mode:true

  }

)

@steps.step0

@steps.step1

**ПРИЛОЖЕНИЕ Б**

**Код программы methods.rb**

class Methods

  def initialize(params = {})

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

  end

  def gcd(a,b)

    puts "using default gcd(#{a}, #{b})" if @debug\_mode

    while b != 0

      remainder = a % b

      a = b

      b = remainder

*#debug*

      sleep 0.5 if @debug\_mode

      puts "gcd(#{a}, #{b})" if @debug\_mode

    end

    return a

  end

  def gcd\_bin(a,b)

    puts "using binary gcd(#{a}, #{b})" if @debug\_mode

    shift = 0

    while a != b

      if a % 2 == 0 && b % 2 == 0

        a = a / 2

        b = b / 2

        shift += 1

      elsif a % 2 == 0

        a = a / 2

      elsif b % 2 == 0

        b = b / 2

      elsif a > b

        a = (a-b)/2

      else

        b = (b-a)/2

      end

*#debug*

      sleep 0.5 if @debug\_mode

      puts "gcd(#{a}, #{b})" if @debug\_mode

    end

    return a \* (2 \*\* shift)

  end

  def gcd\_ext(a, b, first = true)

    puts "using extended gcd(#{a}, #{b})" if @debug\_mode && first

    if a == 0

      return b, 0, 1

    else

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

*#debug*

      sleep 0.5 if @debug\_mode

      puts "gcd(#{a}, #{b}); koeff: (#{x}, #{y})" if @debug\_mode

      return res, y - (b / a) \* x, x

    end

  end

  def inverse(a, md)

    puts %{using inverse of #{a} in #{md}} if @debug\_mode

    gcd, x, \_ = gcd\_ext(a, md)

    puts %{gcd = #{gcd}, x = #{x}} if @debug\_mode

    if gcd != 1

      raise "\nNo inverse element exists\n"

    else

      return x % md

    end

  end

  def chineese\_reminder\_theorem(coefficients = [], modulus = [])

    if coefficients.empty? || modulus.empty?

      raise "Not enough data!"

    end

    puts %{using chineese\_reminder\_theorem for #{coefficients} in #{modulus}} if @debug\_mode

    x = 0

    fact = modulus.reduce(:\*)

    coefficients.zip(modulus).each do |a, m|

      ci = fact / m

      ci\_inv = inverse(ci, m)

      x += a \* ci \* ci\_inv

    end

    x %= fact

    return {x:x, md: fact}

  end

  def exea(a, b)

    return [0, 1] if a % b == 0

    x, y = exea(b, a % b)

    [y, x - y \* (a / b)]

  end

  def mult\_row\_to\_num(row, num, field)

    row.map { |el| (el \* num) % field }

  end

  def add\_rows(row1, row2, field)

    row1.each\_with\_index.map { |el, i| (el + row2[i]) % field }

  end

  def del\_zero\_rows(matrix)

    matrix.reject! { |row| row.all?(&:zero?) }

  end

  def swap\_columns(matrix, col)

    (col + 1...matrix[col].size - 1).each do |i|

      if matrix[col][i] != 0

        matrix.each { |row| row[col], row[i] = row[i], row[col] }

        return

      end

    end

  end

  def gauss(matrix, field)

    matrix.each\_with\_index do |row, i|

      swap\_columns(matrix, i) if row[i] == 0

      rev\_el = exea(row[i], field)[0]

      matrix[i] = mult\_row\_to\_num(row, rev\_el, field)

      matrix.each\_with\_index do |row2, j|

        next if i == j

        matrix[j] = add\_rows(row2, mult\_row\_to\_num(matrix[i], -row2[i], field), field)

      end

    end

    del\_zero\_rows(matrix)

    have\_solution = !matrix.any? { |row| row.last != 0 && row.take(matrix.size).all?(&:zero?) }

    return matrix, have\_solution

  end

  def get\_matrix(matrix)

    matrix.each { |row| puts row.join(' ') }

  end

  def get\_ans(input, field)

    matrix = input[0]

    boo = input[1]

    if !boo

      puts "\n[GAUSS] There are no solutions!"

      return

    end

*# puts "\nAllowed solution of the original system:"*

*# matrix.each do |row|*

*#   row.each\_with\_index do |el, j|*

*#     if j == row.size - 2*

*#       print "#{el}x#{j + 1} = "*

*#     elsif j == row.size - 1*

*#       puts el*

*#     else*

*#       print "#{el}x#{j + 1} + "*

*#     end*

*#   end*

*# end*

    puts "\n[GAUSS] General solution of the original system:"

    matrix.each\_with\_index do |row, i|

      print "x#{i + 1} = "

      (i + 1...row.size - 1).each do |j|

        if matrix[i][j] != 0

          matrix[i][j] = -matrix[i][j] + field

          print "#{matrix[i][j]}x#{j + 1} + "

        end

      end

      puts matrix[i].last

    end

*# print "\n[GAUSS] Free unknowns:"*

*# vals = gets.chomp.split(' ').map(&:to\_i)*

    vals = [1]

    res = []

    matrix.each do |row|

      ans = 0

      (matrix.size...row.size - 1).each { |j| ans += vals[j - matrix.size] \* row[j] }

      ans += row.last

      res << ans

    end

    res.concat(vals)

*# print "\n[GAUSS] Particular solution: ("*

    fin = []

    res.each\_with\_index do |el, i|

      if i == res.size - 1

        fin << el % field

*# print "#{el % field})"*

      else

        fin << el % field

*# print "#{el % field}, "*

      end

    end

    return fin

  end

end

**ПРИЛОЖЕНИЕ В**

**Код программы steps.rb**

require 'matrix'

require './methods.rb'

class Steps

  def initialize(params = {})

    params.dig(:debug\_mode)

  end

  def step0

    puts "Введите размер поля:"

    @p = gets.strip.to\_i

    puts "Ведедите секрет s (s < p):"

    @s = gets.strip.to\_i

    puts "Введите количество сторон n\_max:"

    @n\_max = gets.strip.to\_i

    puts "Введите количество сторон, необходимых для восстановления секрета n\_min:"

    @n\_min = gets.strip.to\_i

    puts "Реализуем (#{@n\_min},#{@n\_max})-пороговое разделение ключа #{@s}!"

    @iv = [@s]

    (@n\_min-1).times {@iv << gen\_rand\_nuber}

    puts "Задали ключевую точку iv:#{@iv}"

    @clients = []

    num = 1

    @n\_max.times do

      ownv = []

      @n\_min.times {ownv << gen\_rand\_nuber}

      ownv << gen\_d(@iv, ownv)

      @clients << {id: num, ownv: ownv}

      num += 1

    end

    puts "Сформированы следующие клиенты:"

    @clients.each {|client| puts client}

  end

  def step1

    puts "\nПроверим восстановление секрета"

    ans = :y

    while ans == :y

      puts "\nВведите id клиентов для восстановления секрета (n1 n2 ...):"

      ids = gets.split.map(&:to\_i)

      matrix = []

      ids.each do |id|

        @clients.each do |client|

          matrix << client[:ownv] if client[:id] == id

          puts "Выбран клиент: #{client}" if client[:id] == id && @debug\_mode

        end

      end

      puts "Для восстановления секрета данными клиентами необходимо решить систему:"

      matrix.each do |line|

        str = "| ( "

        line[0..-2].each\_with\_index do |ai, i|

          str << "#{ai} \* x#{i + 1} + "

        end

        str << "#{line[-1]} ) mod #{@p} = 0"

        puts str

*# line << 0*

      end

      puts "Матрица выглядит следующим образом:"

      pp matrix

      puts "Решим систему методом Гаусса:"

      new\_iv = solve\_gauss\_system(matrix)

      puts "Проверим правильность полученного ключа:"

      all\_good = true

      @iv.each\_with\_index do |a, i|

        puts "iv[#{i}]: #{a},\tnew\_iv[#{i}]: #{new\_iv[i]}"

        all\_good &= a == new\_iv[i]

      end

      puts "\nПроверка ключа #{all\_good ? "пройдена!" : "НЕ пройдена!"}"

      puts "\nВыбрать других клиентов? [y,n]"

      ans = gets.strip.to\_sym

    end

  end

  private

  def gen\_rand\_nuber

    return rand(@p)

  end

  def gen\_d(iv, ownv)

    d = 0

    iv.each\_with\_index do |a, i|

      d += a \* ownv[i]

    end

    d = (-d % @p)

  end

  def solve\_gauss\_system(matrix)

    @methods = Methods.new({debug\_mode: @debug\_mode})

    field = @p

    matrix.each do |line|

      line[-1] = (@p - line[-1]) % @p

    end

*# pp matrix*

    rows, cols = matrix.size, @n\_min + 1

    triangular, status = @methods.gauss(matrix, field)

    puts "\n[GAUSS] Triangular matrix: "

    @methods.get\_matrix(matrix)

    result = @methods.get\_ans([matrix, status], field)

    return result

  end

end