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 (t, n) −пороговое разделение секрета для любых t, n.

Разделяемым секретом в схеме Блэкли является одна из координат точки в m-мерном пространстве. Долями секрета, раздаваемые сторонам, являются уравнения $(m-1)$ −мерных гиперплоскостей. Для восстановления точки необходимо знать m уравнений гиперплоскостей. Менее, чем m сторон не смогут восстановить секрет, так как множеством пересечения $m-1$ плоскостей является прямая, и секрет не может быть восстановлен.

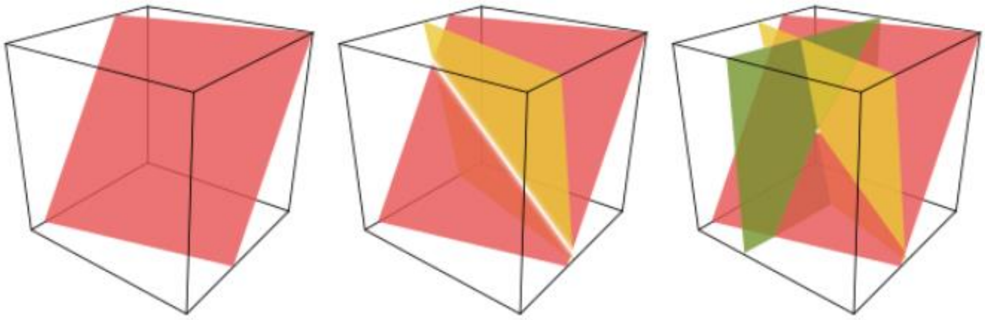

Рисунок 1 - Пример схемы Блэкли в трёх измерениях: каждая доля секрета — это плоскость, а секрет — это одна из координат точки пересечения плоскостей. Двух плоскостей недостаточно для определения точки пересечения.

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 (k, n) −пороговую схему, то есть секрет M разделить между n сторонами так, чтобы любые k из них могли восстановить секрет. Для этого выбирается большое простое число $p > M$, по модулю которого будет строиться поле $GF(p)$. Случайным образом дилер выбирает числа $b_2, b_3, \ldots, b_k \in GF(p)$. Тем самым задается точка $(M, b_2, \ldots, b_k)$ в k −мерном пространстве, первая координата которой является секретом.

Раздача секрета

Для каждой стороны $P_i, i = (1, n)$ случайным образом выбираются коэффициенты $a_{1i}, \ldots, a_{ki}$, равномерно распределённые в поле $GF(p)$. Так как уравнение плоскости имеет вид $a_{li} * x_1 + a_{2i} * x_2 + \cdots + a_{ki} * x_k + d_i = 0$, для каждой стороны необходимо вычислить коэффициенты d_i:

$$d_1 = -(a_{11} * M + a_{21} b_2 + \cdots + a_{k1} * b_k) \bmod p,$$

$$\ldots$$

$$d_i = -(a_{1i} * M + a_{2i} b_2 + \cdots + a_{ki} * b_k) \bmod p,$$

$$\ldots$$

$$d_n = -(a_{1n} * M + a_{2n} b_2 + \cdots + a_{kn} * b_k) \bmod p.$$

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

Восстановление секрета

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

$$\begin{cases} (a_{11} * x_1 + a_{21} x_2 + \cdots + a_{k1} * x_k + d_1) \bmod p = 0, \\ (a_{12} * x_1 + a_{22} x_2 + \cdots + a_{k2} * x_k + d_2) \bmod p = 0, \\ \ldots \\ (a_{1k} * x_1 + a_{2k} x_2 + \cdots + a_{kk} * x_k + d_k) \bmod p = 0. \end{cases}$$

Решение системы даёт точку в k −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 $GF(p)$.

Если число участников встречи будет меньше, чем k, например, $k-1$, то результатом решения системы уравнений, составленной из имеющегося набора коэффициентов, будет прямая в k −мерном пространстве. Тем самым множество допустимых значений секрета, удовлетворяющих полученной системе, в точности совпадает с полным числом элементов поля $GF(p)$, 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.

```
PS F:\all\crypto_protokols\n6> ruby .\go.rb
Введите размер поля:
11
Ведедите секрет s (s < p):
6
Введите количество сторон n_max:
5
Введите количество сторон, необходимых для восстановления секрета n_min:
3
Реализуем (3,5)-пороговое разделение ключа 6!
Задали ключевую точку iv:[6, 6, 9]
Сформированы следующие клиенты:
{:id=>1, :ownv=>[6, 3, 4, 9]}
{:id=>2, :ownv=>[4, 1, 4, 0]}
{:id=>3, :ownv=>[2, 3, 9, 10]}
{:id=>4, :ownv=>[8, 6, 6, 5]}
{:id=>5, :ownv=>[7, 2, 9, 8]}

Проверим восстановление секрета

Введите id клиентов для восстановления секрета (n1 n2 ...):
1 5 2
Для восстановления секрета данными клиентами необходимо решить систему:
| ( 6 * x1 + 3 * x2 + 4 * x3 + 9 ) mod 11 = 0
| ( 7 * x1 + 2 * x2 + 9 * x3 + 8 ) mod 11 = 0
| ( 4 * x1 + 1 * x2 + 4 * x3 + 0 ) mod 11 = 0
Матрица выглядит следующим образом:
[[6, 3, 4, 9], [7, 2, 9, 8], [4, 1, 4, 0]]
Решим систему методом Гаусса:

[GAUSS] Triangular matrix:
1 0 0 6
0 1 0 6
0 0 1 9

[GAUSS] General solution of the original system:
x1 = 6
x2 = 6
x3 = 9
Проверим правильность полученного ключа:
iv[0]: 6,       new_iv[0]: 6
iv[1]: 6,       new_iv[1]: 6
iv[2]: 9,       new_iv[2]: 9

Проверка ключа пройдена!

Выбрать других клиентов? [y,n]
y
```

Рисунок 2 - Пример работы программы

```
Выбрать других клиентов? [y,n]
y

Введите id клиентов для восстановления секрета (n1 n2 ...):
1 5
Для восстановления секрета данными клиентами необходимо решить систему:
| ( 6 * x1 + 3 * x2 + 4 * x3 + 2 ) mod 11 = 0
| ( 7 * x1 + 2 * x2 + 9 * x3 + 3 ) mod 11 = 0
Матрица выглядит следующим образом:
[[6, 3, 4, 2], [7, 2, 9, 3]]
Решим систему методом Гаусса:

[GAUSS] Triangular matrix:
1 0 7 8
0 1 2 9

[GAUSS] General solution of the original system:
x1 = 4x3 + 8
x2 = 9x3 + 9
Проверим правильность полученного ключа:
iv[0]: 6,       new_iv[0]: 1
iv[1]: 6,       new_iv[1]: 7
iv[2]: 9,       new_iv[2]: 1

Проверка ключа НЕ пройдена!
```

Рисунок 3 - Пример работы программы

ПРИЛОЖЕНИЕ А
Код программы go.rb

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

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

@steps.step0
@steps.step1
```

ПРИЛОЖЕНИЕ Б
Код программы methods.rb

```ruby
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
```

```ruby
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

```ruby
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|
```

```ruby
        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
системму:"
    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
```

```ruby
  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
```