

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

**Протоколы открытого распределения ключей**

**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

\_\_\_\_\_

подпись, дата

**Р. А. Фарахутдинов**

Саратов 2023

## **1 Постановка задачи**

### Цель работы:

- Изучение протокола открытого распределения ключей Диффи-Хеллмана и его программная реализация.

### Задачи работы:

- Изучить протокол Диффи-Хеллмана, его сильные и слабые стороны;
- Привести программную реализацию протокола.

## 2 Теоретические сведения

Протокол Диффи — Хеллмана (англ. Diffie–Hellman key exchange protocol, DH) — криптографический протокол, позволяющий двум и более сторонам получить общий секретный ключ, используя незащищенный от прослушивания канал связи. Полученный ключ используется для шифрования дальнейшего обмена с помощью алгоритмов симметричного шифрования.

Схема открытого распределения ключей, предложенная Диффи и Хеллманом, произвела настоящую революцию в мире шифрования, так как снимала основную проблему классической криптографии — проблему распределения ключей.

В чистом виде алгоритм Диффи — Хеллмана уязвим для модификации данных в канале связи, в том числе для атаки «Man-in-the-middle (человек посередине)», поэтому схемы с его использованием применяют дополнительные методы односторонней или двусторонней аутентификации.

Работу алгоритма можно описать следующим образом. Предположим, существует два абонента: Алиса и Боб. Обоим абонентам известны некоторые два числа  $g$  и  $p$ , которые не являются секретными и могут быть известны также другим заинтересованным лицам. Для того, чтобы создать неизвестный более никому секретный ключ, оба абонента генерируют большие случайные числа: Алиса — число  $a$ , Боб — число  $b$ . Затем Алиса вычисляет остаток от деления:

$$A = g^a \bmod p$$

и пересылает его Бобу, а Боб вычисляет остаток от деления

$$B = g^b \bmod p$$

и передаёт Алисе. Предполагается, что злоумышленник может получить оба этих значения, но не модифицировать их (то есть, у него нет возможности вмешаться в процесс передачи).

На втором этапе Алиса на основе имеющегося у неё  $a$  и полученного по сети  $B$  вычисляет значение:

$$B^a \bmod p = g^{ab} \bmod p$$

Боб на основе имеющегося у него  $b$  и полученного по сети  $A$  вычисляет значение

$$A^b \bmod p = g^{ab} \bmod p$$

Как нетрудно видеть, у Алисы и Боба получилось одно и то же число:

$$K = g^{ab} \bmod p$$

Его они и могут использовать в качестве секретного ключа, поскольку здесь злоумышленник встретится с практически неразрешимой (за разумное время) проблемой вычисления ключа по перехваченным  $g^a \bmod p$  и  $g^b \bmod p$ , если числа  $p, a, b$  выбраны достаточно большими. Работа алгоритма показана на Рисунке 1.

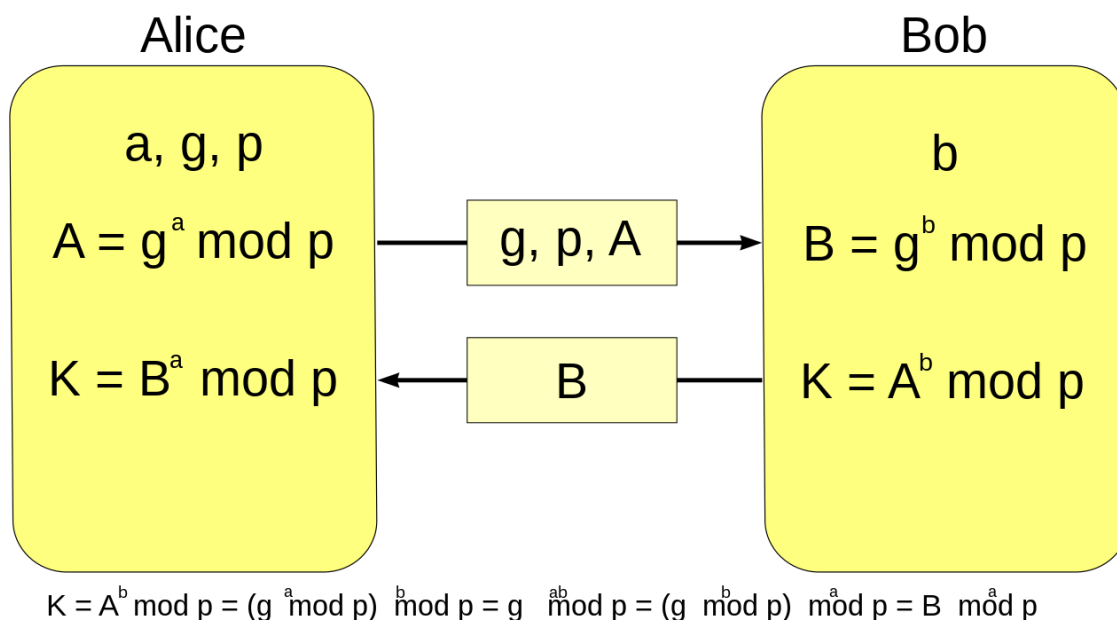


Рисунок 1 - Схема работы алгоритма Диффи-Хеллмана

Опишем алгоритм Диффи-Хеллмана формально.

Общие параметры:  $p$  – большое простое число,  $g$  – примитивный корень по модулю  $p$ .

Протокол:

1.  $A \rightarrow B: \{X = g^x \bmod p\}$ , где  $x$  – случайное секретное целое число Алисы из интервала  $1 < x < p$ .
2.  $B \rightarrow A: \{Y = g^y \bmod p\}$ , где  $y$  – случайное секретное целое число Боба из интервала  $1 < y < p$ .

3.  $A: K = Y^x \bmod p, B: K = X^y \bmod p$  (Алиса и Боб независимо друг от друга вычисляют их сеансовый ключ  $K$ ).

### 3 Тестирование программы

На рисунках 2-3 представлены результаты работы программы, эмулирующие оба прохода протокола Диффи-Хеллмана между двумя абонентами сети. В ходе работы программы пользователь вводит режим работы программы (*y* – с отображением промежуточных шагов, *n* – без отображения промежуточных шагов), а также битовую длину случайно генерируемых простых чисел.

```
PS C:\Users\smallany\Desktop\crypto_protocols\n1> ruby .\proc.rb
Enable debug? y/n
n

Enter binary length for prime number
15

--- DIFFIE-HELLMAN STARTS ---

Alice and Bob applying prime numbers(g, n)

Applied: (g,n) = (15161, 5563)

Alice forms own secret number: 12948

Bob forms own secret number: 5472

Bob forms half-key: 2825
Alice gets half-key from Bob : 2825

Alice forms half-key: 123
Bob gets half-key from Alice : 123

Alice calculates secret key

Alice's secret key: K = 2602

Bob calculates secret key

Bob's secret key: K = 2602

--- DIFFIE-HELLMAN ENDS ---

Alice's final state:
{:name=>"Alice", :g=>15161, :n=>5563, :secret=>12948, :half_key=>2825, :secret_key=>2602}

Bob's final state:
{:name=>"Bob", :g=>15161, :n=>5563, :secret=>5472, :half_key=>123, :secret_key=>2602}

PS C:\Users\smallany\Desktop\crypto_protocols\n1> █
```

Рисунок 2 - Работа программы

```

PS C:\Users\smallpany\Desktop\crypto_protokols\n1> ruby .\proc.rb
Enable debug? y/n
y

Enter binary length for prime number
4

--- DIFFIE-HELLMAN STARTS ---

Alice and Bob applying prime numbers(g, n)

[RAND] Generating random int:
[BIN] 0111
[INT] 7
[PRIME?] Test for 7 is prime: true

[RAND] Generating random int:
[BIN] 1011
[INT] 11
[PRIME?] Test for 11 is prime: true

Applied: (g,n) = (7, 11)

[RAND] Generating random int:
[BIN] 0001
[INT] 1

Alice forms own secret number: 1

[RAND] Generating random int:
[BIN] 0010
[INT] 2

Bob forms own secret number: 2

[POW] pow(7, 2) mod 11 = 5

Bob forms half-key: 5
Alice gets half-key from Bob : 5

[POW] pow(7, 1) mod 11 = 7

Alice forms half-key: 7
Bob gets half-key from Alice : 7

Alice calculates secret key

[POW] pow(5, 1) mod 11 = 5

Alice's secret key: K = 5

Bob calculates secret key

[POW] pow(7, 2) mod 11 = 5

Bob's secret key: K = 5

--- DIFFIE-HELLMAN ENDS ---

Alice's final state:
{:name=>"Alice", :g=>7, :n=>11, :secret=>1, :half_key=>5, :secret_key=>5}

Bob's final state:
{:name=>"Bob", :g=>7, :n=>11, :secret=>2, :half_key=>7, :secret_key=>5}

PS C:\Users\smallpany\Desktop\crypto_protokols\n1> █

```

Рисунок 3 - Работа программы (с отображением промежуточных действий)

## ПРИЛОЖЕНИЕ А

### Код программы proc.rb

```
require './methods.rb'

puts %{Enable debug? y/n}
@debug_mode = gets.strip == 'y'

puts %{\nEnter binary length for prime number}
@len = gets.strip.to_i

@methods = Methods.new({debug_mode: @debug_mode})

def get_prime
  # puts %{\nEnter binary length for prime number}
  # len = gets.strip.to_i
  # len = 15

  first_num = 0

  while ! @methods.is_prime(first_num)
    first_num = @methods.gen_random_int(@len)
  end

  first_num
end

def get_int
  # puts %{\nEnter binary length for random secret number}
  # len = gets.strip.to_i
  # len = 10

  @methods.gen_random_int(@len)
end

def apply_gn
  puts %{\nAlice and Bob applying prime numbers(g, n)}
  open_keys = {g: get_prime, n: get_prime}
  puts %{\nApplied: (g,n) = ({open_keys[:g]}, #{open_keys[:n]})}

  open_keys
end

def get_secret_number(client)

  own_secret = get_int

  puts %{\n#{client[:name]} forms own secret number: #{own_secret}}

  own_secret
end

def get_half_key(source, destination)

  half_key = @methods.powm(source[:g], source[:secret], source[:n])

  puts %{\n#{source[:name]} forms half-key: #{half_key}}
```



```

        puts %#{destination[:name]} gets half-key from #{source[:name]} :
        #{half_key}}

        half_key
    end

    def enum_secret_key(client)
        puts %{\n#{client[:name]} calculates secret key}

        secret_key = @methods.powm(client[:half_key], client[:secret],
        client[:n])

        puts %{\n#{client[:name]}'s secret key: K = #{secret_key}}

        secret_key
    end

    def show_client(client)
        puts %{\n#{client[:name]}'s final state:}
        pp client
    end

    def diffie_hellman

        puts %{\n--- DIFFIE-HELLMAN STARTS ---\n}

        open_keys = apply_gn

        a_client = {name: 'Alice', g: open_keys[:g], n: open_keys[:n]}
        b_client = {name: 'Bob', g: open_keys[:g], n: open_keys[:n]}

        a_client[:secret] = get_secret_number(a_client)
        b_client[:secret] = get_secret_number(b_client)

        a_client[:half_key] = get_half_key(b_client, a_client)
        b_client[:half_key] = get_half_key(a_client, b_client)

        a_client[:secret_key] = enum_secret_key(a_client)
        b_client[:secret_key] = enum_secret_key(b_client)

        puts %{\n--- DIFFIE-HELLMAN ENDS ---\n}

        show_client(a_client)
        show_client(b_client)

        puts
    end

    diffie_hellman

```

## ПРИЛОЖЕНИЕ Б

### Код программы methods.rb

```
class Methods

  def initialize(params = {})
    @debug_mode = params.dig(:debug_mode)
  end

  def gen_random_int(length = 0)

    raise 'Not enough number length!' if length == 0

    # srand(Time.now.to_f)

    bin_str = ''
    num = 0

    while num == 0
      length.times do
        # srand(Time.now.to_f)
        bin_str += rand(2).to_s
      end

      num = bin_str.to_i(2)
    end

    if @debug_mode
      puts %{\n\t[RAND] Generating random int:}
      sleep 0.5
      puts %{\t[BIN] #{bin_str}\n\t[INT] #{num}}
    end

    num
  end

  def is_prime(number = 0)
    test = ("1" * number) !~ /^1?$(11+?)\1+$/

    puts %{\t[PRIME?] Test for #{number} is prime: #{test}} if
@debug_mode && number != 0
    test
  end

  def powm(key = 0, degree, md)
    raise 'Empty key!' if key == 0

    result = 1

    degree.times do |e|
      result = (result * key) % md
    end

    puts %{\n\t[POW] pow(#{key}, #{degree}) mod #{md} = #{result}}
  end
end
```

```
        result
    end
end
```