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 обмена ключами**

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 протокола обмена ключами Нидхема-Шрёдера и его программная реализация.

Задачи работы:

* Изучить протокол Нидхема-Шрёдера, его сильные и слабые стороны;
* Привести программную реализацию протокола.

**2 Теоретические сведения**

Протокол для аутентификации с симметричным ключом, вероятно являющийся самым знаменитым протоколом аутентификации и установления ключа, был сформулирован Майклом Шрёдером и Роджером Нидхемом в 1978 году. Однако, он уязвим для атаки, изобретенной Дороти Деннинг (англ. Dorothy E. Denning) и Джованни Марией Сакко (англ. Giovanni Maria Sacco) в 1981 году. Несмотря на это, он стал основой для целого класса подобных протоколов. В частности, протокол Kerberos является одним из вариантов Нидхем-Шрёдер-протокола аутентификации на основе доверенной третьей стороны и его модификациях, предложенных Деннинг и Сакко. Протокол Нидхема-Шрёдера для аутентификации с открытым ключом также является уязвимым. В 1995 году Лоу (англ. Gavin Lowe) описал возможную атаку на протокол.

При схеме шифрования с симметричным ключом, предполагается, что секретный ключ известен и серверу аутентификации (Трент) и обоим субъектам обмена: (Алиса) и (Боб). Изначально оба субъекта имеют секретные ключи: , известные только им и некоторой доверенной стороне — серверу аутентификации. В ходе выполнения протокола Алиса и Боб получают от сервера новый секретный сессионный ключ для шифрования взаимных сообщений в данном сеансе связи, то есть сообщения от Алисы к Бобу расшифровать может только Боб, сообщения от Боба к Алисе расшифровать может только Алиса. Кроме того, субъекты обмена должны быть уверены, что пришедшее сообщение было отправлено именно тем, с кем должен произойти обмен. Боб должен быть уверен, что получил сообщение именно от Алисы и наоборот. Это также обеспечивается протоколом. Предположим, что обмен инициирует Алиса. Будем полагать, что сервер аутентификации у них общий. Рассмотрим реализацию протокола:

Обмен начинается с того, что Алиса генерирует некоторое случайное число (идентификатор), использующееся один раз. Первое сообщение от Алисы к Тренту содержит в себе имена участников предстоящего обмена и генерированное Алисой случайное число:

Данное сообщение посылается открытым текстом, но может быть зашифровано ключом Алисы :

При получении этого сообщения Трент извлекает из базы данных секретные ключи Алисы и Боба: , а также вычисляет новый сессионный ключ . Далее Трент посылает Алисе следующее сообщение:

Алиса может расшифровать и прочесть сообщение от Трента. Она проверяет наличие своего идентификатора в сообщении, что подтверждает то, что данное сообщение является откликом на её первое сообщение Тренту. Также она проверяет имя субъекта, с которым собирается обмениваться данными. Эта проверка обязательна, так как если бы не было этого имени, Злоумышленник мог бы заменить имя Боба на своё в первом сообщении, и Алиса, ничего не подозревая, в дальнейшем бы взаимодействовала со Злоумышленником. Часть сообщения Алиса прочитать не может, так как эта часть зашифрована ключом Боба. Алиса пересылает Бобу зашифрованный его ключом фрагмент:

Расшифровать его может только Боб, так как оно зашифровано его секретным ключом. После расшифровки Боб тоже владеет сессионным ключом . Имя Алисы в сообщении подтверждает факт, что сообщение от неё. Далее при обмене данными будет использоваться сессионный ключ. Чтобы сделать схему симметричной и уменьшить вероятность атаки воспроизведения, Боб генерирует некоторое случайное число (идентификатор Боба) и посылает Алисе следующее сообщение, зашифрованное сессионным ключом:

Алиса расшифрует его и посылает отклик, который ожидает Боб, также зашифрованный сессионным ключом:

Для регулярно взаимодействующих партнёров можно сократить число сообщений до трёх, убрав первые два. При этом ключ будет использоваться многократно.

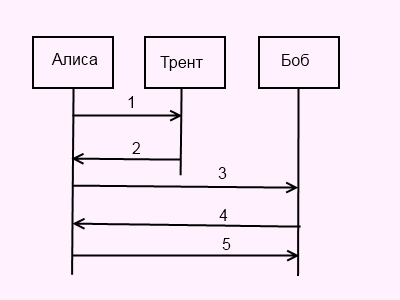


Рисунок 1 - Протокол Нидхема-Шрёдера для аутентификации с симметричным ключом

**3 Тестирование программы**

На рисунках 2-3 представлены результаты работы программы, эмулирующей работу протокола Нидхема-Шрёдера между сервером аутентификации (Trent) и двумя субъектами обмена (Alice и Bob).

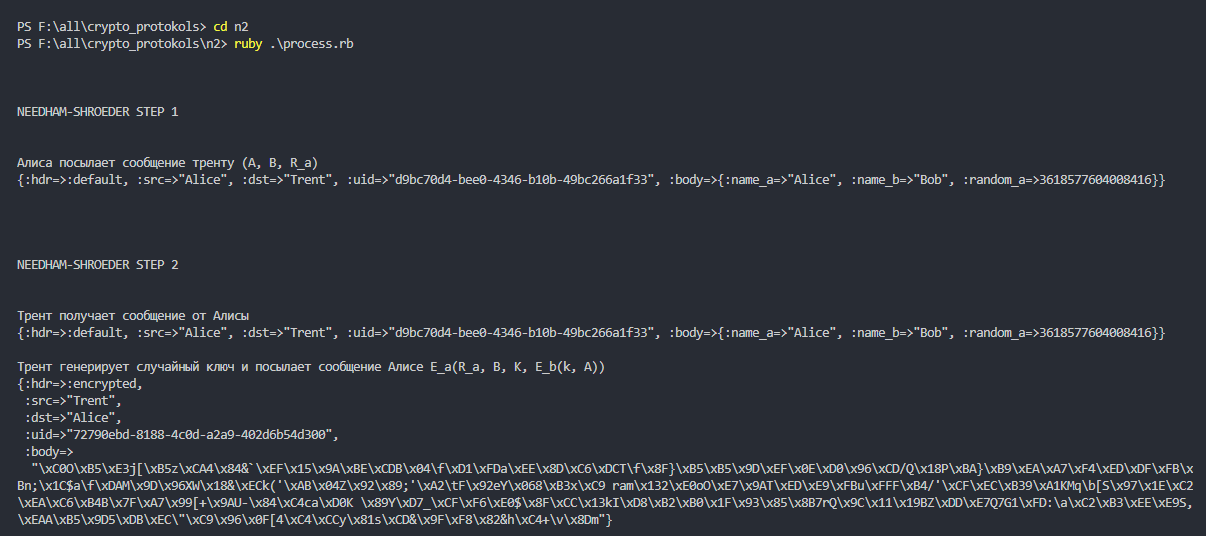


Рисунок 2 - Работа программы

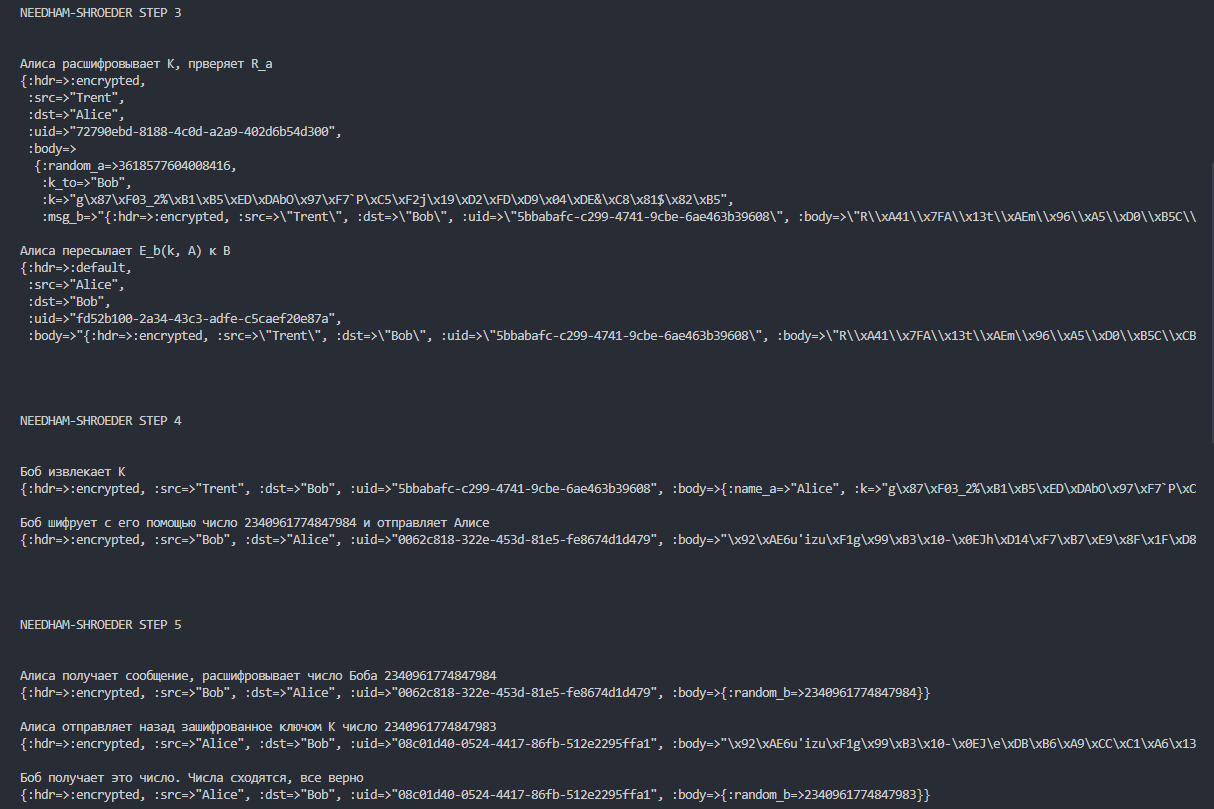


Рисунок 3 - Работа программы

**ПРИЛОЖЕНИЕ А**

**Код программы methods.rb**

class Methods

  def initialize(params = {})

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

    @bin\_length = params.dig(:bin\_length).to\_i

  end

  def gen\_random\_int()

    raise 'Not enough number length!' if @bin\_length == 0

    bin\_str = '1'

    (@bin\_length - 1).times do

      bin\_str += rand(2).to\_s

    end

    num = bin\_str.to\_i(2)

    if @debug\_mode == :all

      puts %{\n\t[RAND] Generating random int:}

      sleep 0.5

      puts %{\t[BIN] #{bin\_str}\n\t[INT] #{num}}

    end

    num

  end

end

**ПРИЛОЖЕНИЕ Б**

**Код программы needham\_shroeder.rb**

require './methods.rb'

require 'encryption'

require './names.rb'

require 'securerandom'

require 'digest'

require 'json'

class NeedhamShroeder

  def initialize(params = {})

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

    @methods = Methods.new(params.dig(:methods\_params))

    @encryptor = Encryption::Symmetric.new

    @encryptor.iv = SecureRandom.random\_bytes(16)

  end

  def start

    kk = gen\_random\_key

*#Инициализация клиентов*

    a\_client = make\_client('Alice')

    b\_client = make\_client('Bob')

    t\_client = make\_client('Trent')

*#по условию Алиса и Трент, Боб и Трент уже имеют общий ключ*

    bt\_session\_key = gen\_random\_key

    at\_session\_key = gen\_random\_key

    a\_client[:session\_keys][t\_client[:name].to\_sym] = at\_session\_key

    t\_client[:session\_keys][a\_client[:name].to\_sym] = at\_session\_key

    b\_client[:session\_keys][t\_client[:name].to\_sym] = bt\_session\_key

    t\_client[:session\_keys][b\_client[:name].to\_sym] = bt\_session\_key

*#Алиса посылает сообщение тренту (A, B, R\_a)*

    msg = {

      name\_a: a\_client[:name],

      name\_b: b\_client[:name],

      random\_a: a\_client[:own\_secret]

    }

    send(a\_client, t\_client, msg, :default)

    puts "\n\n\nNEEDHAM-SHROEDER STEP 1\n\n"

    puts "\nАлиса посылает сообщение тренту (A, B, R\_a)"

    pp a\_client[:pushed][-1]

    gets

*#Трент генерирует случайный ключ и посылает сообщение Алисе E\_a(R\_a, B, K, E\_b(k, A))*

    t\_client[:decrypted] << process\_pulled(t\_client)

    lst\_msg = t\_client[:decrypted][-1][:body]

    new\_key = "AB session key"

    body\_to\_b = {

      name\_a: lst\_msg[:name\_a],

      k: new\_key

    }

    @msg\_to\_b = make\_message(t\_client, b\_client, body\_to\_b, :encrypted)

    msg\_to\_a = {

      random\_a: lst\_msg[:random\_a],

      k\_to: lst\_msg[:name\_b],

      k: new\_key,

      msg\_b: @msg\_to\_b.to\_s

    }

    send(t\_client, a\_client, msg\_to\_a, :encrypted)

    a\_client[:decrypted] << process\_pulled(a\_client)

    puts "\n\n\nNEEDHAM-SHROEDER STEP 2\n\n"

    puts "\nТрент получает сообщение от Алисы"

    pp t\_client[:decrypted][-1]

    puts "\nТрент генерирует случайный ключ и посылает сообщение Алисе E\_a(R\_a, B, K, E\_b(k, A))"

    pp t\_client[:pushed][-1]

    gets

*#Алиса расшифровывает К, прверяет R\_a и пересылает E\_b(k, A) к В*

    lst\_msg = a\_client[:decrypted][-1][:body]

    raise "R\_a not matched! " if a\_client[:own\_secret] != lst\_msg[:random\_a]

    a\_client[:session\_keys][lst\_msg[:k\_to].to\_sym] = kk

    a\_client[:decrypted][-1][:body][:k] = kk

    send(a\_client, b\_client, lst\_msg[:msg\_b], :default)

    puts "\n\n\nNEEDHAM-SHROEDER STEP 3\n\n"

    puts "\nАлиса расшифровывает К, прверяет R\_a"

    pp a\_client[:decrypted][-1]

    puts "\nАлиса пересылает E\_b(k, A) к В"

    pp a\_client[:pushed][-1]

    gets

*#Боб извлекает К, шифрует с его помощью число и отправляет алисе*

    b\_client[:decrypted] << process\_pulled(b\_client, :body)

    lst\_msg = b\_client[:decrypted][-1][:body]

    b\_client[:decrypted][-1][:body][:k] = kk

    b\_client[:session\_keys][lst\_msg[:name\_a].to\_sym] = kk

    msg = {

      random\_b: b\_client[:own\_secret]

    }

    send(b\_client, a\_client, msg, :encrypted)

    puts "\n\n\nNEEDHAM-SHROEDER STEP 4\n\n"

    puts "\nБоб извлекает К"

    pp b\_client[:decrypted][-1]

    puts "\nБоб шифрует с его помощью число #{b\_client[:own\_secret]} и отправляет Алисе"

    pp b\_client[:pushed][-1]

    gets

    a\_client[:decrypted] << process\_pulled(a\_client)

    number = a\_client[:decrypted][-1][:body][:random\_b]

    msg = {

      random\_b: number - 1

    }

    send(a\_client, b\_client, msg, :encrypted)

    b\_client[:decrypted] << process\_pulled(b\_client)

    puts "\n\n\nNEEDHAM-SHROEDER STEP 5\n\n"

    puts "\nАлиса получает сообщение, расшифровывает число Боба #{number}"

    pp a\_client[:decrypted][-1]

    puts "\nАлиса отправляет назад зашифрованное ключом K число #{number - 1}"

    pp a\_client[:pushed][-1]

    puts "\nБоб получает это число. Числа сходятся, все верно"

    pp b\_client[:decrypted][-1]

    gets

    puts "\n\n[Alice final state]"

    pp a\_client

    gets

    puts "\n\n[Bob final state]"

    pp b\_client

    gets

    puts "\n\n[Trent final state]"

    pp t\_client

    gets

    0

  end

  private

  def make\_client(name = {})

    {

      name: name || NAMES.sample,

      pushed: [],

      pulled: [],

      decrypted: [],

      own\_secret: SecureRandom.rand(10\*\*16),

      session\_keys: {}

    }

  end

  def make\_message(src, dst, body, mode)

    case mode

    when :encrypted

      enc\_key = src[:session\_keys][dst[:name].to\_sym]

      enc\_body = encrypt\_message(enc\_key, body)

      {

        hdr: mode,

        src: src[:name],

        dst: dst[:name],

        uid: SecureRandom.uuid,

*# time: Time.now,*

        body: enc\_body

      }

    when :default

      {

        hdr: mode,

        src: src[:name],

        dst: dst[:name],

        uid: SecureRandom.uuid,

*# time: Time.now,*

        body: body

      }

    else

      {

        hdr: :error

      }

    end

  end

  def send(src, dst, body, mode)

    src[:pushed] << make\_message(src, dst, body, mode)

    dst[:pulled] << src[:pushed].last

    [src, dst]

  end

  def process\_pulled(client, mode = :default)

    packet = (mode == :default) ? client[:pulled][-1] : @msg\_to\_b

*# pp packet*

    case packet[:hdr]

    when :encrypted

      body = decrypt\_message(client[:session\_keys][packet[:src].to\_sym], packet[:body])

      {

        hdr: packet[:hdr],

        src: packet[:src],

        dst: packet[:dst],

        uid: packet[:uid],

*# time: packet[:time],*

        body: body

      }

    when :default

      packet

    else

      {

        hdr: :error\_in\_decryption

      }

    end

  end

  def encrypt\_message(key, message)

    @encryptor.key = key

    @encryptor.encrypt (hash\_to\_string(message))

  end

  def decrypt\_message(key, message)

    @encryptor.key = key

    string\_to\_hash(@encryptor.decrypt(message))

  end

  def hash\_to\_string(hash)

*# pp hash*

    JSON.dump(hash)

  end

  def string\_to\_hash(string)

    JSON.parse(string).transform\_keys(&:to\_sym)

  end

  def gen\_random\_key

    SecureRandom.random\_bytes(32)

  end

end

**ПРИЛОЖЕНИЕ В**

**Код программы process.rb**

require './needham\_shroeder.rb'

params = {

  debug\_mode: 'all',

  methods\_params: {

    debug\_mode: 'all',

    bin\_length: 10

  }

}

ns = NeedhamShroeder.new(params)

# pp ns

ns.start