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 $T$ (Трент) и обоим субъектам обмена: $A$ (Алиса) и $B$ (Боб). Изначально оба субъекта имеют секретные ключи: $K_A$, $K_B$, 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:

1. $Alice \rightarrow A, B, R_A \rightarrow Trent$
2. $Trent \rightarrow \left\{R_A, B, K, \{K, A\}_{K_B}\right\}_{K_A} \rightarrow Alice$
3. $Alice \rightarrow \{K, A\}_{K_B} \rightarrow Bob$

4. $Bob \rightarrow \{R_B\}_K \rightarrow Alice$
5. $Alice \rightarrow \{R_b - 1\}_k \rightarrow Bob$

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

$$Alice \rightarrow A, B, R_A \rightarrow Trent$$

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

$$Alice \rightarrow \{A, B, R_A\}_{K_A} \rightarrow Trent$$

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

$$Trent \rightarrow \left\{R_A, B, K, \{K, A\}_{K_B}\right\}_{K_A} \rightarrow Alice$$

Алиса может расшифровать и прочесть сообщение от Трента. Она проверяет наличие своего идентификатора $R_A$ 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:

$$Alice \rightarrow \{K, A\}_{K_B} \rightarrow Bob$$

Расшифровать его может только Боб, так как оно зашифровано его секретным ключом. После расшифровки Боб тоже владеет сессионным ключом $K$. 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 $R_B$ (идентификатор Боба) и посылает Алисе следующее сообщение, зашифрованное сессионным ключом:

$$Bob \rightarrow \{R_B\}_K \rightarrow Alice$$

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

$$Alice \rightarrow \{R_b - 1\}_k \rightarrow Bob$$

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

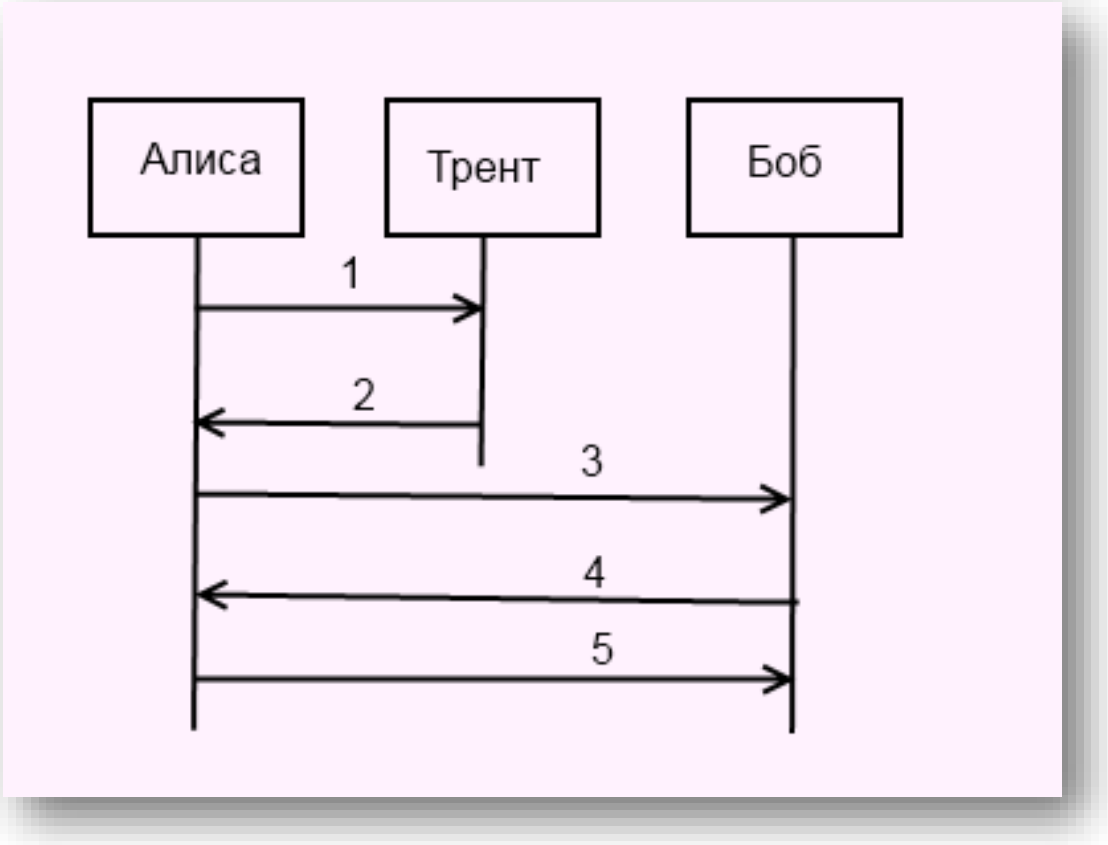


Рисунок 1 - Протокол Нидхема-Шрёдера для аутентификации с симметричным ключом

## 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).

```
PS F:\all\crypto_protokols> cd n2
PS F:\all\crypto_protokols\n2> ruby .\process.rb



NEEDHAM-SHROEDER STEP 1


Алиса посылает сообщение тренту (A, B, R_a)
{:hdr=>:default, :src=>"Alice", :dst=>"Trent", :uid=>"d9bc70d4-bee0-4346-b10b-49bc266a1f33", :body=>{:name_a=>"Alice", :name_b=>"Bob", :random_a=>3618577604008416}}




NEEDHAM-SHROEDER STEP 2


Трент получает сообщение от Алисы
{:hdr=>:default, :src=>"Alice", :dst=>"Trent", :uid=>"d9bc70d4-bee0-4346-b10b-49bc266a1f33", :body=>{:name_a=>"Alice", :name_b=>"Bob", :random_a=>3618577604008416}}

Трент генерирует случайный ключ и посылает сообщение Алисе E_a(R_a, B, K, E_b(k, A))
{:hdr=>:encrypted,
 :src=>"Trent",
 :dst=>"Alice",
 :uid=>"72790ebd-8188-4c0d-a2a9-402d6b54d300",
 :body=>
  "\xC0O\xB5\xE3j[\xB5z\xCA4\x84&`\xEF\x15\x9A\xBE\xCDB\x04\f\xD1\xFDa\xEE\x8D\xC6\xDCT\f\x8F}\xB5\xB5\x9D\xEF\x0E\xD0\x96\xCD/Q\x18P\xBA}\xB9\xEA\xA7\xF4\xED\xDF\xFB\x
Bn;\x1C$a\f\xDAM\x9D\x96XW\x188\xECk('\xAB\x04Z\x92\x89;'\xA2\tF\x92eY\x068\xB3x\xC9 ram\x132\xE0o0\xE7\x9AT\xED\xE9\xFBu\xFFF\xB4/'\xCF\xEC\xB39\xA1KMq\b[S\x97\x1E\xC2
\xEA\xC6\xB4B\x7F\xA7\x99[+\x9AU-\x84\xC4ca\xD0K \x89Y\xD7_\xCF\xF6\xE0$\x8F\xCC\x13kI\xD8\xB2\xB0\x1F\x93\x85\x8B7rQ\x9C\x11\x19BZ\xDD\xE7Q7G1\xFD:\a\xC2\xB3\xEE\xE9S,
\xEAA\xB5\x9D5\xDB\xEC\"\xC9\x96\x0F[4\xC4\xCCy\x81s\xCD&\x9F\xF8\x82&h\xC4+\v\x8Dm"}
```

Рисунок 2 - Работа программы

```
NEEDHAM-SHROEDER STEP 3


Алиса расшифровывает K, прверяет R_a
{:hdr=>:encrypted,
 :src=>"Trent",
 :dst=>"Alice",
 :uid=>"72790ebd-8188-4c0d-a2a9-402d6b54d300",
 :body=>
  {:random_a=>3618577604008416,
   :k_to=>"Bob",
   :k=>"g\x87\xF03_2%\xB1\xB5\xED\xDAbO\x97\xF7`P\xC5\xF2j\x19\xD2\xFD\xD9\x04\xDE&\xC8\x81$\x82\xB5",
   :msg_b=>"{:hdr=>:encrypted, :src=>\"Trent\", :dst=>\"Bob\", :uid=>\"5bbabafc-c299-4741-9cbe-6ae463b39608\", :body=>\"R\\xA41\\x7FA\\x13t\\xAEm\\x96\\xA5\\xD0\\xB5C\\

Алиса пересылает E_b(k, A) к B
{:hdr=>:default,
 :src=>"Alice",
 :dst=>"Bob",
 :uid=>"fd52b100-2a34-43c3-adfe-c5caef20e87a",
 :body=>"{:hdr=>:encrypted, :src=>\"Trent\", :dst=>\"Bob\", :uid=>\"5bbabafc-c299-4741-9cbe-6ae463b39608\", :body=>\"R\\xA41\\x7FA\\x13t\\xAEm\\x96\\xA5\\xD0\\xB5C\\xCB




NEEDHAM-SHROEDER STEP 4


Боб извлекает K
{:hdr=>:encrypted, :src=>"Trent", :dst=>"Bob", :uid=>"5bbabafc-c299-4741-9cbe-6ae463b39608", :body=>{:name_a=>"Alice", :k=>"g\x87\xF03_2%\xB1\xB5\xED\xDAbO\x97\xF7`P\xC

Боб шифрует с его помощью число 2340961774847984 и отправляет Алисе
{:hdr=>:encrypted, :src=>"Bob", :dst=>"Alice", :uid=>"0062c818-322e-453d-81e5-fe8674d1d479", :body=>"\x92\xAE6u'izu\xF1g\x99\xB3\x10-\x0EJh\xD14\xF7\xB7\xE9\x8F\x1F\xD8




NEEDHAM-SHROEDER STEP 5


Алиса получает сообщение, расшифровывает число Боба 2340961774847984
{:hdr=>:encrypted, :src=>"Bob", :dst=>"Alice", :uid=>"0062c818-322e-453d-81e5-fe8674d1d479", :body=>{:random_b=>2340961774847984}}

Алиса отправляет назад зашифрованное ключом K число 2340961774847983
{:hdr=>:encrypted, :src=>"Alice", :dst=>"Bob", :uid=>"08c01d40-0524-4417-86fb-512e2295ffa1", :body=>"\x92\xAE6u'izu\xF1g\x99\xB3\x10-\x0EJ\e\xDB\xB6\xA9\xCC\xC1\xA6\x13

Боб получает это число. Числа сходятся, все верно
{:hdr=>:encrypted, :src=>"Alice", :dst=>"Bob", :uid=>"08c01d40-0524-4417-86fb-512e2295ffa1", :body=>{:random_b=>2340961774847983}}
```

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 K, прверяет R_a и пересылает E_b(k, A) к B
    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 K, прверяет R_a"
    pp a_client[:decrypted][-1]
    puts "\nАлиса пересылает E_b(k, A) к B"
    pp a_client[:pushed][-1]
    gets

    #Боб извлекает K, 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 K"
    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
```