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 анонимности

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 анонимности Sensus и его программная реализация.

Задачи работы:

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

2 Теоретические сведения

Схема Фудзиоки-Окамото-Оты, разработанная в 1992 году, основывается на протоколе двух агентств и криптографической подписи вслепую. Несильно усложняя протокол, эта схема частично решает проблему сговора двух агентств. Для работы протокола необходим заранее выбранный способ маскирующего шифрования, под которым избиратель присылает регистратору бюллетень. Ослепляющее (маскирующее) шифрование — особый вид шифрования, позволяющее удостовериться в том, что документ подлинный и подписан авторизованным пользователем, но не даёт узнать содержащиеся в нём данные. Маскирующее шифрование должно быть коммутативным с электронной подписью, то $sign\big(blind(B)\big) = blind(sign(B))$.

Введем следующие обозначения:

- A — агентство, проводящее электронное голосование;
- E — избиратель, легитимный участник голосования;
- B — цифровой бюллетень;
- V — регистратор.

Алгоритм протокола Фудзиоки - Окамото - Оты:

Шаг 1. V утверждает списки легитимных избирателей

Шаг 2. E выполняет следующие действия:

- создаёт $e_{public}, e_{private}$ (для цифровой подписи), e_{secret} (для того, чтобы ни A, ни посторонний злоумышленник не мог до нужного времени узнать содержимое бюллетеня);
- подготавливает сообщение B с выбранным решением;
- шифрует его с помощью e_{secret};
- накладывает слой ослепляющего шифрования;
- подписывает его с помощью $e_{private}$;
- отправляет V $blind(sign(e_{private}, encrypt(e_{secret}, B)))$.

Шаг 3. V выполняет следующие действия:

- создаёт v_{public} и $v_{private}$, публичный ключ выкладывается в общий доступ;
- удостоверяется, что бюллетень действительный и принадлежит легитимному и не голосовавшему избирателю;
- подписывает его с помощью $v_{private}$;
- возвращает его E.

Шаг 4. E снимает с бюллетени слой маскирующего шифрования (в силу коммутативности остаётся $sign(v_{private}, sign(e_{private}, encrypt(e_{secret}, B)))$ и отправляет ее A;

Шаг 5. A выполняет следующие действия:

- проверяет подписи E и V;
- помещает всё ещё зашифрованную e_{secret} бюллетень в специальный список, который будет опубликован после того, как все избиратели проголосуют или по истечении заранее оговорённого срока.

Шаг 6. После того как список появляется в открытом доступе, E высылает A e_{secret}.

Шаг 7. A выполняет следующие действия:

- расшифровывает сообщение;
- подсчитывает результаты.

Лорри Кранор и Рон Ситрон (англ. Lorrie Faith Cranor, Ron K. Cytron) в 1996 предложили модификацию протокола Фудзиоки — Окамото — Оты под названием Sensus. Отличие заключается в шагах 5-6. После того, как A получило зашифрованное сообщение от E, оно не только добавляет его в публикуемый список, а вдобавок отправляет подписанный бюллетень обратно избирателю в качестве квитанции. Таким образом E не нужно ждать, пока проголосуют все остальные, и он может закончить голосование за один сеанс.

Это не только удобно для конечного пользователя, но ещё и предоставляет дополнительного доказательство, что E участвовал в выборах. Кроме того, в Sensus регламентированы дополнительные вспомогательные модули, упрощающие и автоматизирующие ход голосования.

3 Тестирование программы

На рисунках 1-9 представлены результаты работы программы, эмулирующей проведение голосования, используя протокол Sensus. В качестве входных параметров передается количество избирателей и количество кандидатов (голоса избирателей генерируются случайным образом).

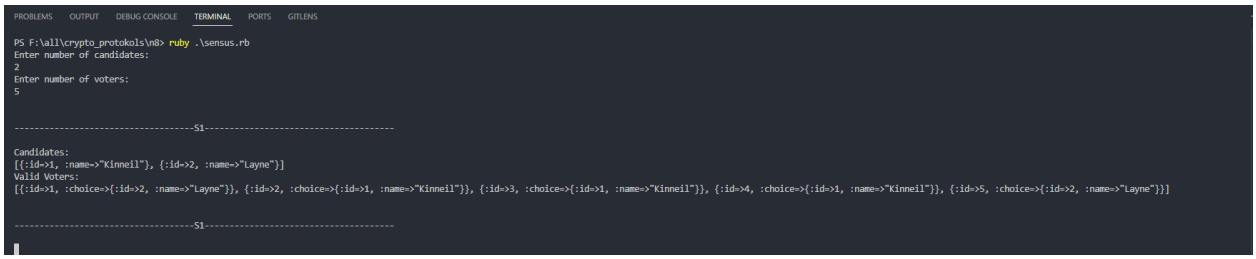

Рисунок 1 - Случайная генерация кандидатов, избирателей и их голосов (5 избирателей, 2 кандидата)

Рисунок 2 - Генерация ключей избирателей, формирование сообщения для Регистратора

Рисунок 3 - Генерация ключей Регистратора, проверка подписей избирателей и подписывание валидных бюллетеней (часть 1)

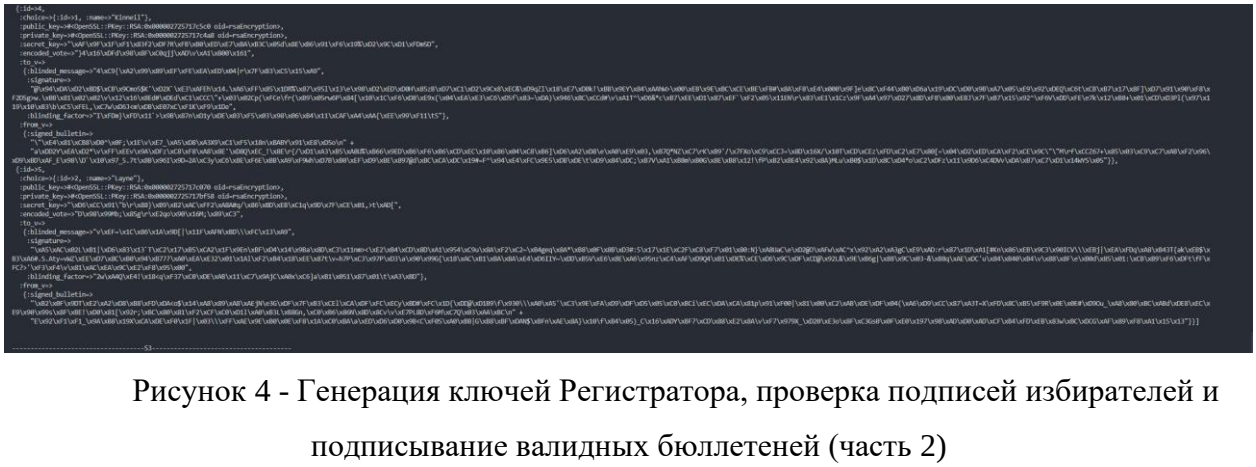

Рисунок 4 - Генерация ключей Регистратора, проверка подписей избирателей и подписывание валидных бюллетеней (часть 2)

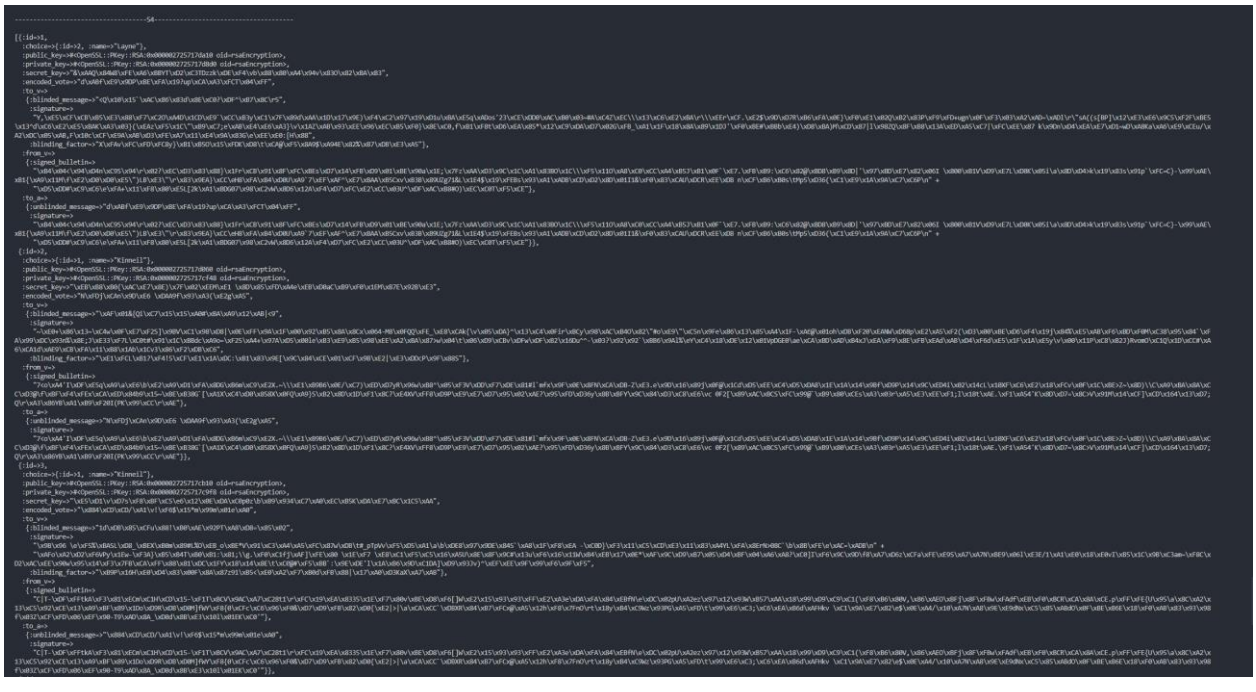

Рисунок 5 - Снятие ослепляющего шифрования избирателями, отправка подписанных Регистратором бюллетеней в Агентство (часть 1)

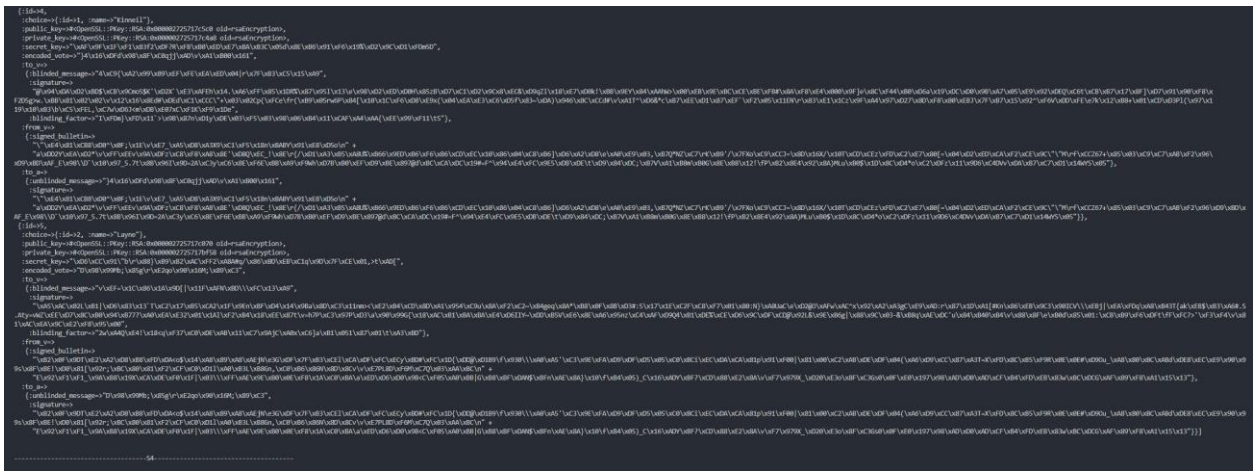

Рисунок 6 - Снятие ослепляющего шифрования избирателями, отправка подписанных Регистратором бюллетеней в Агентство (часть 2)

Рисунок 7 - Набор ключей, идентификаторов и зашифрованных голосов у Агентства после приема бюллетеней от избирателей, подсчет голосов

Рисунок 8 - Вывод результата голосования

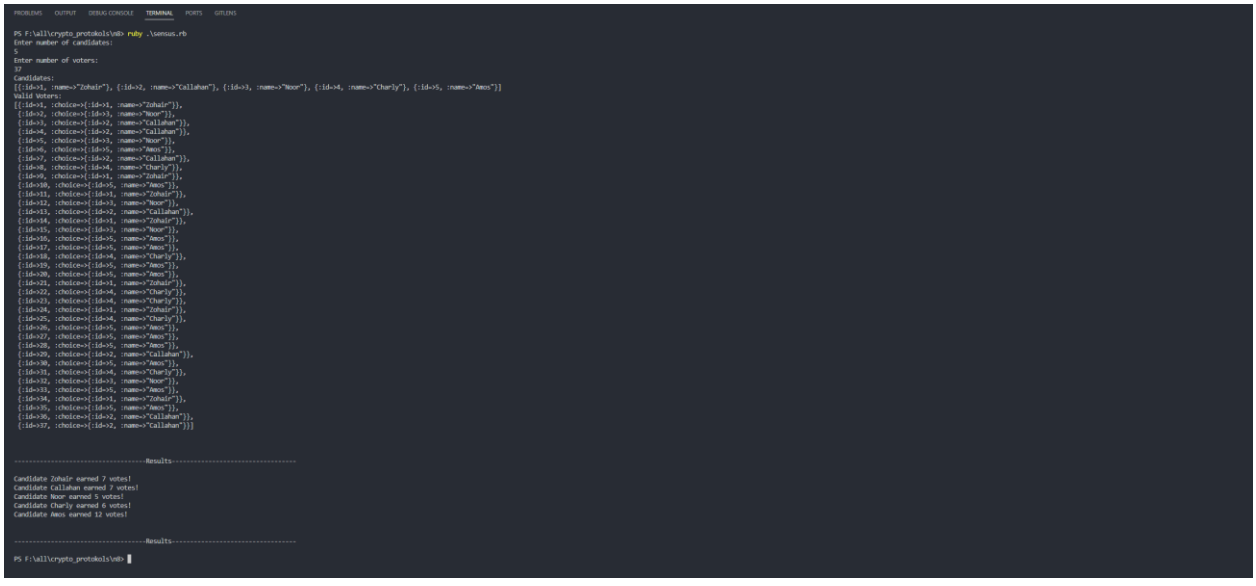

Рисунок 9 - Упрощенный вывод процесса голосования (5 кандидатов, 17 избирателей)

ПРИЛОЖЕНИЕ А
Код программы sensus.rb

```ruby
require 'openssl'
require 'json'
require 'base64'
require './names.rb'

class Sensus
  attr_reader :voters_count, :candidate_count, :candidates, :voters

  def initialize(voters_count, candidate_count, debug_mode)
    @voters_count = voters_count
    @candidate_count = candidate_count
    @candidates = []
    @voters = []
    @register = {
      v_public: OpenSSL::PKey::RSA.generate(2048).public_key,
      v_secret: OpenSSL::PKey::RSA.generate(2048)
    }
    @voting_center = {
      ok_votes: [],
      vote_results: {}
    }

    @debug_mode = debug_mode
  end

  def step1

    @candidates = (1..@candidate_count).map do |id|
      { id: id, name: generate_random_name }
    end


    @voters = (1..@voters_count).map do |id|
      { id: id, choice: @candidates.sample }
    end

    if @debug_mode
      puts "\n\n-----------------------------------S1-------------------------
-----------\n\n"
      puts "Candidates:"
      pp @candidates
      puts "Valid Voters:"
      pp @voters
      puts "\n\n-----------------------------------S1-------------------------
-----------\n\n"
      gets
    else
```

```
      puts "Candidates:"
      pp @candidates
      puts "Valid Voters:"
      pp @voters
      gets
    end
  end

  def step2
    @voters.each do |voter|

      voter[:public_key] = OpenSSL::PKey::RSA.generate(2048).public_key
      voter[:private_key] = OpenSSL::PKey::RSA.generate(2048)
      voter[:secret_key] = OpenSSL::Cipher.new('AES-256-CFB').encrypt.random_key


      candidate_id = voter[:choice][:id]
      message = { candidate_id: candidate_id }.to_json


      encrypted_message = OpenSSL::Cipher.new('AES-256-CFB').encrypt
      encrypted_message.key = voter[:secret_key]
      encrypted_message_text = encrypted_message.update(message) +
encrypted_message.final
      voter[:encoded_vote] = encrypted_message_text


      blinding_factor = OpenSSL::Cipher.new('AES-256-CFB').encrypt.random_key
      blinded_message_bytes = encrypted_message_text.bytes.map.with_index {
|byte, index| byte ^ blinding_factor.bytes[index] }


      blinded_message = blinded_message_bytes.pack('C*')


      signature = voter[:private_key].sign(OpenSSL::Digest::SHA256.new,
blinded_message)


      voter[:to_v] = {
        blinded_message: blinded_message,
        signature: signature,
        blinding_factor: blinding_factor
      }
    end

    if @debug_mode
      puts "\n\n-----------------------------------S2-------------------------
-----------\n\n"
      pp @voters
```

```
      puts "\n\n-----------------------------------S2-------------------------
-----------\n\n"
      gets
    end
  end

  def step3
    voted = []
    @voters.each do |voter|
      next if voted.include?(voter[:id])


      bulletin = {
        blinded_message: voter[:to_v][:blinded_message],
        signature: voter[:to_v][:signature],
        blinding_factor: voter[:to_v][:blinding_factor]
      }


      encoded_bulletin = {
        blinded_message: Base64.strict_encode64(bulletin[:blinded_message]),
        signature: Base64.strict_encode64(bulletin[:signature]),
        blinding_factor: Base64.strict_encode64(bulletin[:blinding_factor])
      }


      if verify_bulletin(encoded_bulletin, voter[:private_key].public_key)

        signed_bulletin = @register[:v_secret].sign(OpenSSL::Digest::SHA256.new,
JSON.generate(encoded_bulletin, :ascii_only => true))

        voter[:from_v] = { signed_bulletin: signed_bulletin }
      else

        voter[:from_v] = nil
      end

      voted << voter[:id]
    end

    if @debug_mode
      puts "\n\n-----------------------------------S3-------------------------
-----------\n\n"
      pp @voters
      puts "\n\n-----------------------------------S3-------------------------
-----------\n\n"
      gets
    end
  end

  def step4
```

```
    @voters.each do |voter|
      next if voter[:from_v].nil?

      unblinded_message_bytes =
voter[:to_v][:blinded_message].bytes.map.with_index { |byte, index| byte ^
voter[:to_v][:blinding_factor].bytes[index] }

      unblinded_message = unblinded_message_bytes.pack('C*')

      voter[:to_a] = {
        unblinded_message: unblinded_message,
        signature: voter[:from_v][:signed_bulletin]
      }
    end

    if @debug_mode
      puts "\n\n-------------------------------------S4-------------------------
-----------\n\n"
      pp @voters
      puts "\n\n-------------------------------------S4-------------------------
-----------\n\n"
      gets
    end
  end

  def step5
    @voters.each do |voter|
      next if voter[:to_a].nil?

      if verify_signature(voter[:to_a][:unblinded_message],
voter[:to_a][:signature], voter[:to_a][:unblinded_message])

        @voting_center[:ok_votes] << {message: voter[:encoded_vote], voter_id:
voter[:id]}

        send_secret_key_to_center(voter[:id], voter[:secret_key])
      end
    end

    decrypt_and_count_votes

    if @debug_mode
      puts "\n\n-------------------------------------S5-------------------------
-------\n\n"
      pp @voting_center
      puts "\n\n-------------------------------------S5-------------------------
-------\n\n"
      gets
    end

    publish_results
```

```ruby
  end

  private

  def generate_random_name
    NAMES.sample
  end

  def verify_bulletin(bulletin, public_key)
    decoded_bulletin = {
      blinded_message: Base64.strict_decode64(bulletin[:blinded_message]),
      signature: Base64.strict_decode64(bulletin[:signature]),
      blinding_factor: Base64.strict_decode64(bulletin[:blinding_factor])
    }

    public_key.verify(OpenSSL::Digest::SHA256.new, decoded_bulletin[:signature],
decoded_bulletin[:blinded_message])
  end

  def verify_signature(message, signature, public_key)
    decoded_message = Base64.decode64(message)
    decoded_signature = Base64.decode64(signature)

    register_verification =
@register[:v_public].verify(OpenSSL::Digest::SHA256.new, decoded_signature,
decoded_message)

    voter_verification = !@voters.any? do |voter|
      next if voter[:to_a].nil?

      voter_private_key = voter[:private_key]
      voter[:to_a][:unblinded_message] == decoded_message &&
        voter_private_key.verify(OpenSSL::Digest::SHA256.new, decoded_signature,
decoded_message)
    end

    # pp register_verification
    # pp voter_verification

    register_verification || voter_verification
  end

  def send_secret_key_to_center(voter_id, private_key)
    @voting_center[:voter_keys] ||= {}
    @voting_center[:voter_keys][voter_id] = private_key
  end

  def decrypt_encoded_vote(encoded_vote, secret_key)
    decipher = OpenSSL::Cipher.new('AES-256-CFB')
    decipher.decrypt
    decipher.key = secret_key
```

```
    decrypted_message = decipher.update(encoded_vote) + decipher.final
    decrypted_message
  end

  def decrypt_and_count_votes
    @candidates.each {|man| @voting_center[:vote_results][man[:id]] = 0}

    @voting_center[:ok_votes].each do |vote|
      # pp vote[:message]
      # pp @voting_center[:voter_keys][vote[:voter_id]]
      res = decrypt_encoded_vote(vote[:message],
@voting_center[:voter_keys][vote[:voter_id]])
      voting = JSON.parse(res).transform_keys!(&:to_sym)
      @voting_center[:vote_results][voting[:candidate_id]] += 1
    end
  end

  def publish_results
    puts "\n\n-----------------------------------Results-----------------------
----------\n\n"
    @candidates.each do |candy|
      puts "Candidate #{candy[:name]} earned
#{@voting_center[:vote_results][candy[:id]]} votes!"
    end
    puts "\n\n-----------------------------------Results-----------------------
----------\n\n"
  end

  def valid_json?(json_str)
    JSON.parse(json_str)
    return true
  rescue JSON::ParserError
    return false
  end
end


puts "Enter number of candidates:"
cd = gets.strip.to_i
puts "Enter number of voters:"
vt = gets.strip.to_i

ss = Sensus.new(vt, cd, false)
ss.step1
ss.step2
ss.step3
ss.step4
ss.step5
```