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 .

Введем следующие обозначения:

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

**Алгоритм протокола Фудзиоки - Окамото - Оты:**

Шаг 1. утверждает списки легитимных избирателей

Шаг 2. выполняет следующие действия:

* создаёт (для цифровой подписи), (для того, чтобы ни A, ни посторонний злоумышленник не мог до нужного времени узнать содержимое бюллетеня);
* подготавливает сообщение с выбранным решением;
* шифрует его с помощью ;
* накладывает слой ослепляющего шифрования;
* подписывает его с помощью ;
* отправляет .

Шаг 3. выполняет следующие действия:

* создаёт и , публичный ключ выкладывается в общий доступ;
* удостоверяется, что бюллетень действительный и принадлежит легитимному и не голосовавшему избирателю;
* подписывает его с помощью ;
* возвращает его .

Шаг 4. снимает с бюллетени слой маскирующего шифрования (в силу коммутативности остаётся и отправляет ее ;

Шаг 5. выполняет следующие действия:

* проверяет подписи и ;
* помещает всё ещё зашифрованную 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, высылает .

Шаг 7. 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 получило зашифрованное сообщение от , оно не только добавляет его в публикуемый список, а вдобавок отправляет подписанный бюллетень обратно избирателю в качестве квитанции. Таким образом 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 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).

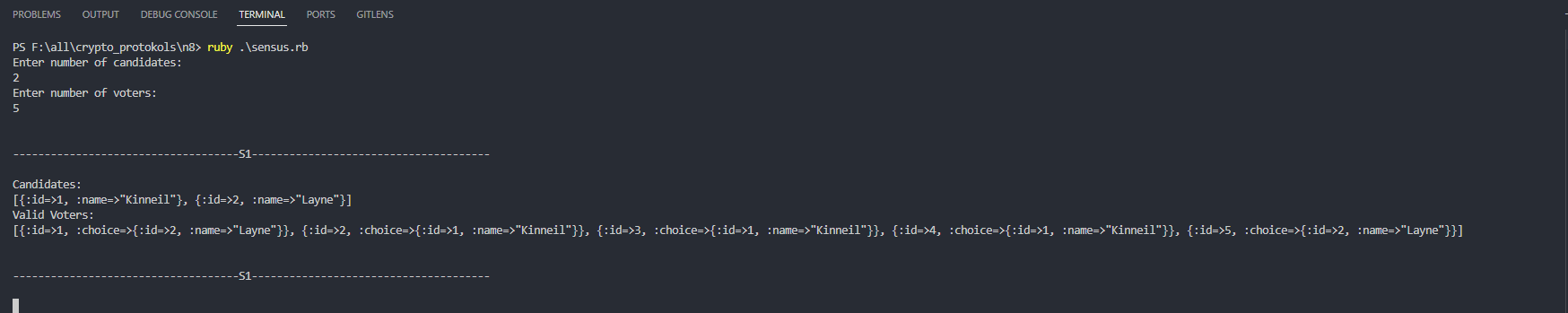


Рисунок 1 - Случайная генерация кандидатов, избирателей и их голосов (5 избирателей, 2 кандидата)

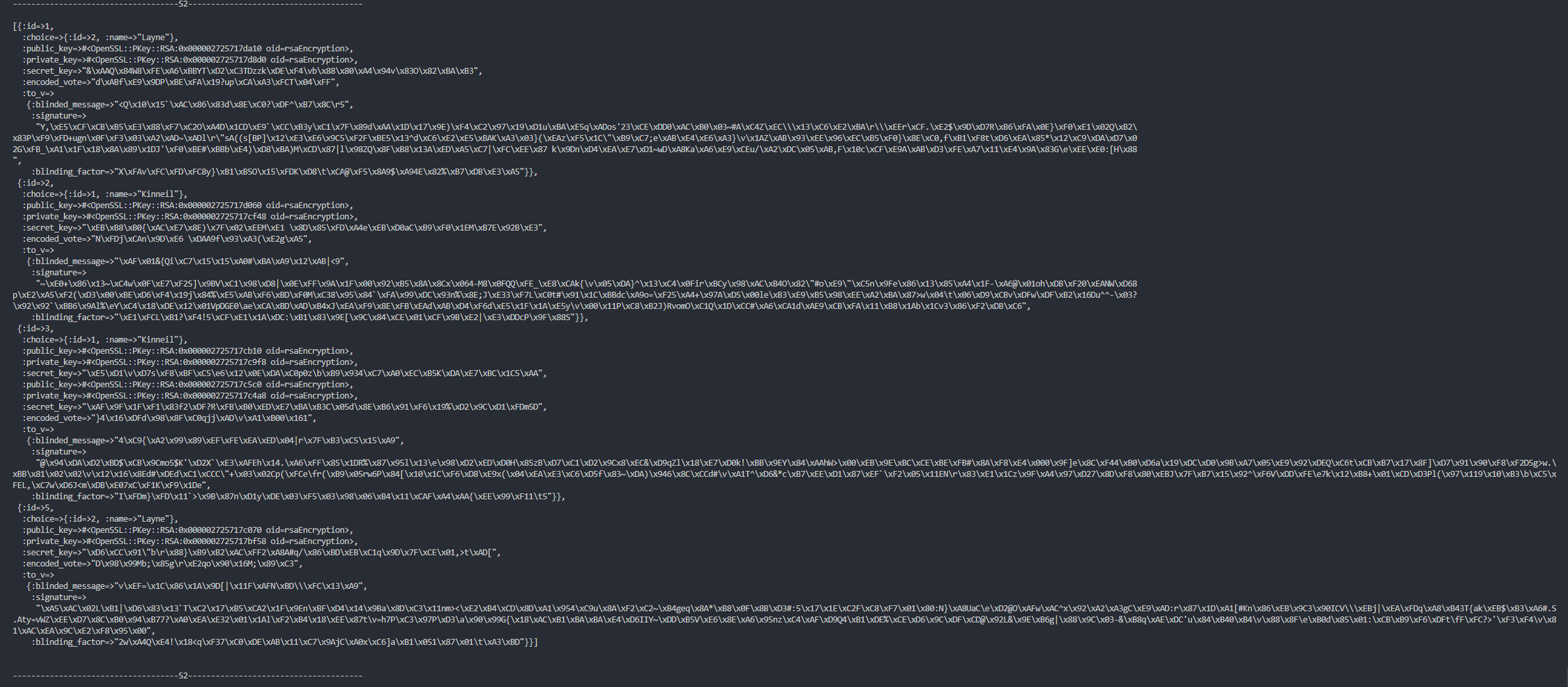


Рисунок 2 - Генерация ключей избирателей, формирование сообщения для Регистратора

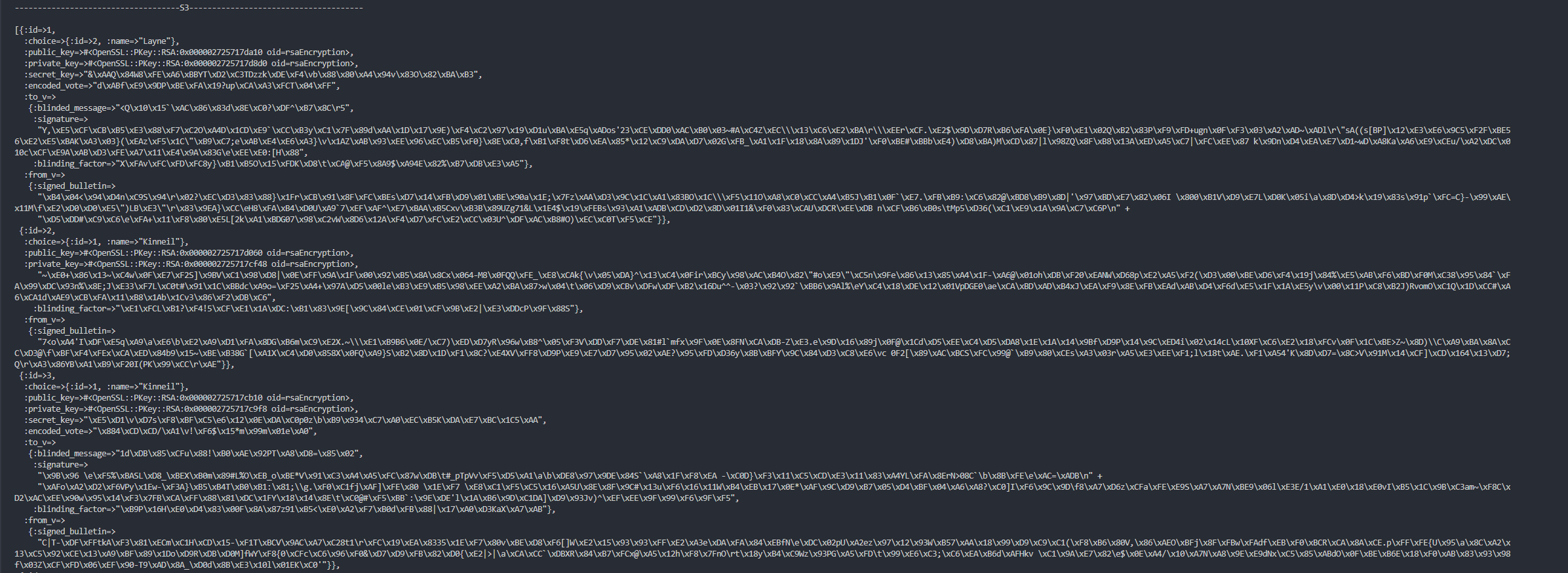


Рисунок 3 - Генерация ключей Регистратора, проверка подписей избирателей и подписывание валидных бюллетеней (часть 1)

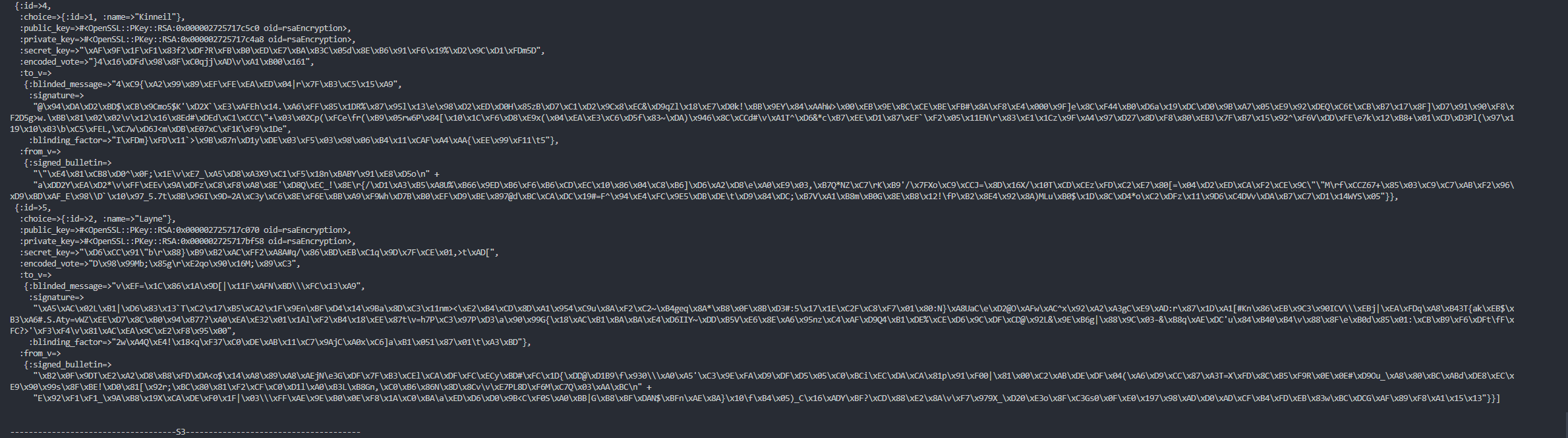


Рисунок 4 - Генерация ключей Регистратора, проверка подписей избирателей и подписывание валидных бюллетеней (часть 2)

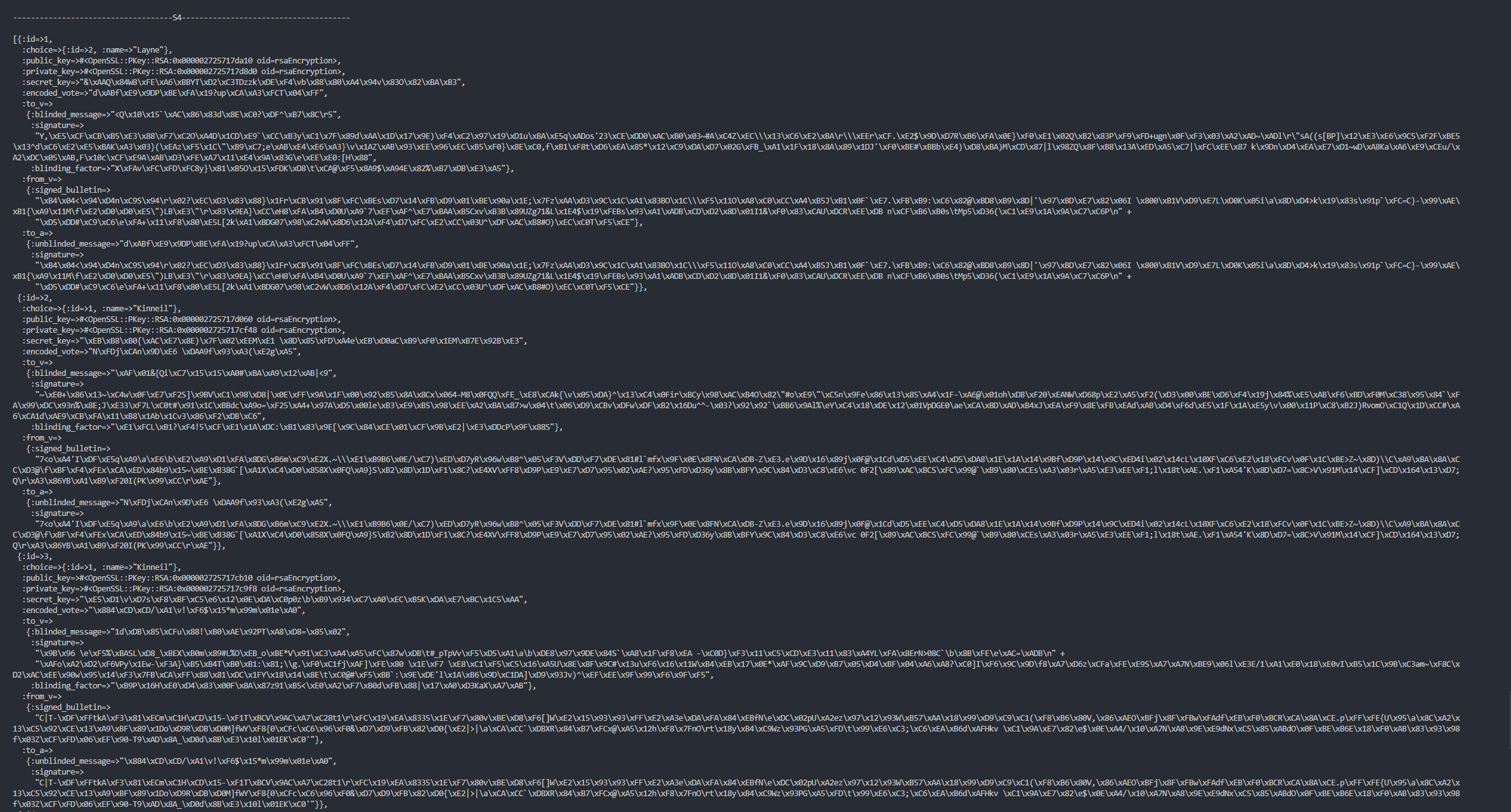


Рисунок 5 - Снятие ослепляющего шифрования избирателями, отправка подписанных Регистратором бюллетеней в Агентство (часть 1)

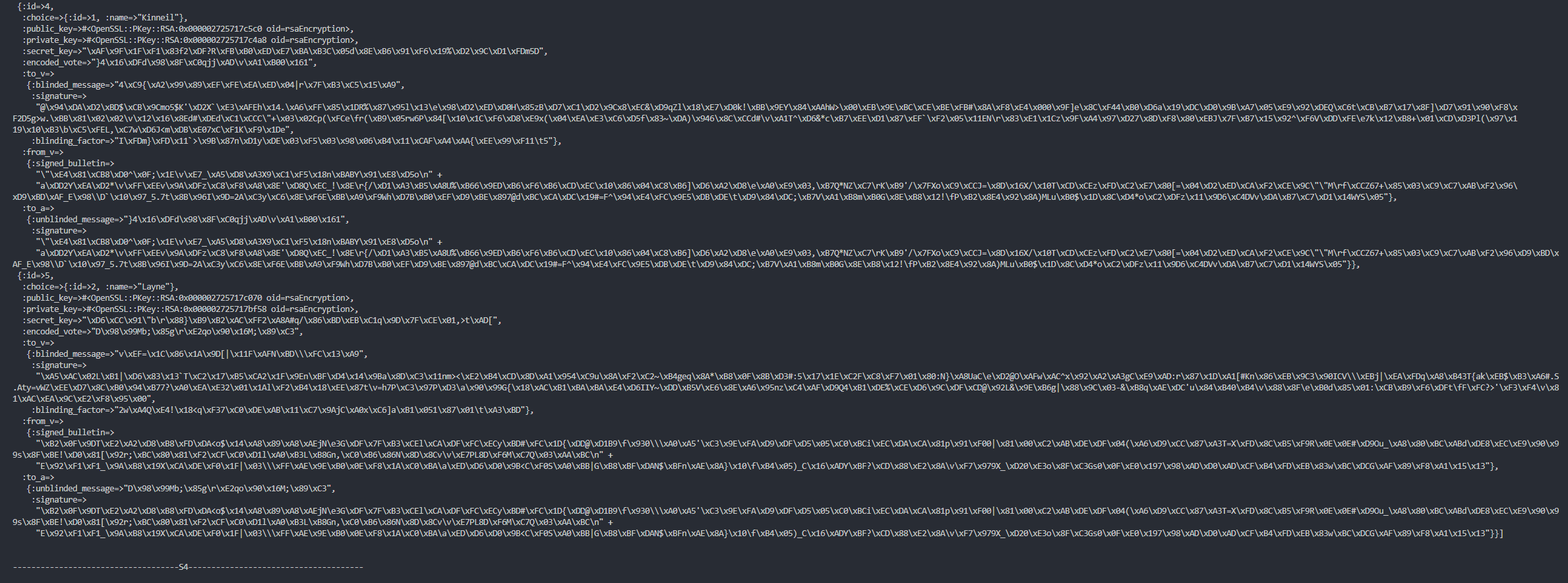


Рисунок 6 - Снятие ослепляющего шифрования избирателями, отправка подписанных Регистратором бюллетеней в Агентство (часть 2)

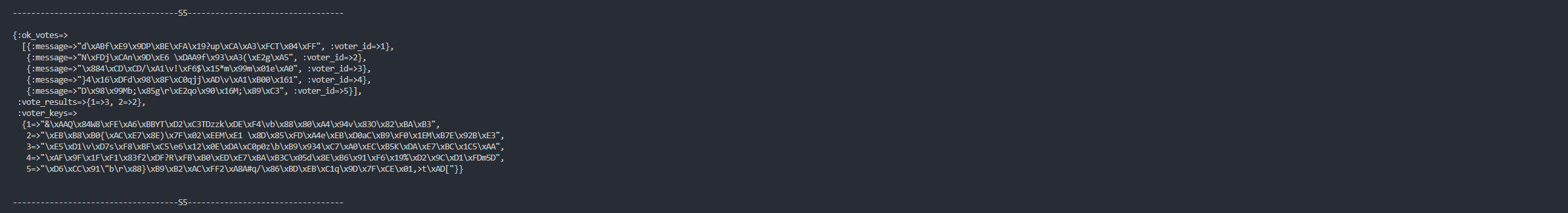


Рисунок 7 - Набор ключей, идентификаторов и зашифрованных голосов у Агентства после приема бюллетеней от избирателей, подсчет голосов

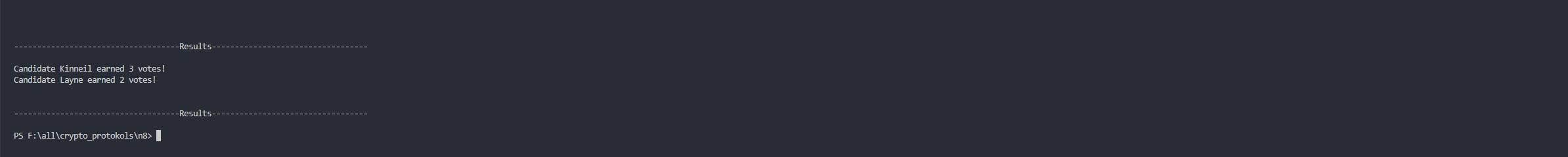


Рисунок 8 - Вывод результата голосования

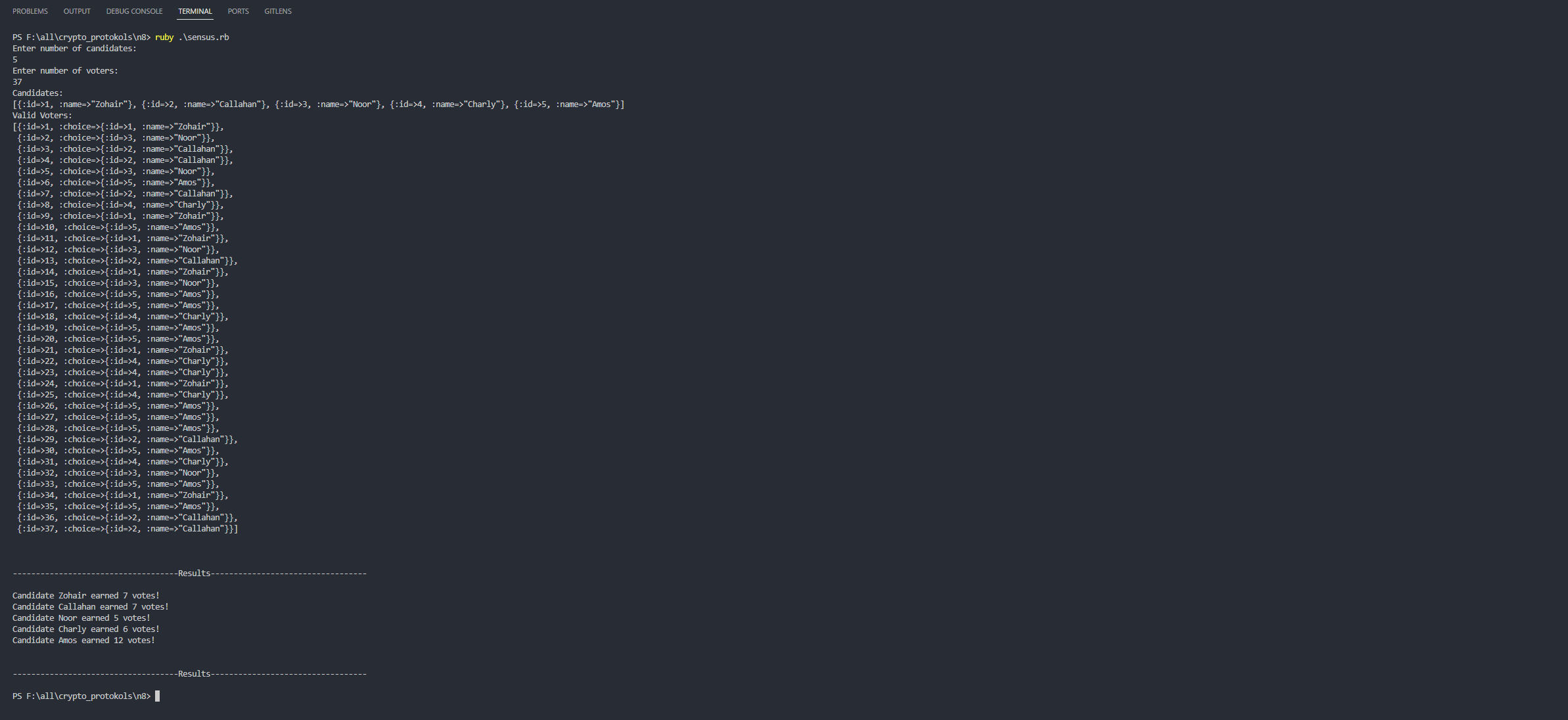


Рисунок 9 - Упрощенный вывод процесса голосования (5 кандидатов, 17 избирателей)

**ПРИЛОЖЕНИЕ А**

**Код программы sensus.rb**

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

  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