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
Бородина Артёма Горовича

Преподаватель
аспирант _____ Р. А. Фарахутдинов
подпись, дата

Саратов 2023

1 Постановка задачи

Необходимо реализовать протокол анонимности Нурми-Саломаа-Сантина.

2 Теоретические сведения

Для начала рассмотрим простой протокол тайного цифрового голосования.

Простой алгоритм электронного голосования по сути представляет собой переписку с электронными подписями между избирательным комитетом и множеством избирателей. Пусть здесь и далее: A – агентство, проводящее электронное голосование, E – избиратель, легитимный участник голосования, B – цифровой бюллетень. B может содержать число, имя кандидата, развёрнутый текст или какие-либо другие данные, сообщающие о выборе E, верифицирующие его или необходимые для усиления безопасности протокола. Ход голосования выглядит так:

1. A выкладывает списки возможных избирателей.
2. Пользователи, в числе которых и E, сообщают о желании участвовать в голосовании.
3. A выкладывает списки легитимных избирателей.

Шаги 1-3 обязательны. Основная цель – определение и объявление числа активных участников n. Хотя некоторые из них могут не участвовать, а некоторые – и вовсе не существовать («мёртвые души», злонамеренно внесённые A), возможность манипулирования голосованием у A заметно снижена. В дальнейшем эти шаги будут считаться за один шаг «утвердить списки».

4. A создаёт открытый и закрытый ключ a_{public} и $a_{private}$ и выкладывает в общий доступ a_{public}. Кто угодно может зашифровать сообщение при помощи a_{public}, но расшифровать его сможет только A.
5. E

- создаёт собственные публичный и приватный ключи ЭЦП e_{public} и $e_{private}$, затем публикует открытый ключ. Кто угодно может проверить документ E, но подписать его – только сам избиратель. Этот шаг пропускается, если A уже знает электронные подписи избирателей (например, они были сгенерированы при регистрации в системе)
- формирует сообщение B, где тем или иным способом выражает свою волю
- подписывает сообщение личным закрытым ключом $e_{private}$
- шифрует сообщение открытым ключом a_{public}
- отправляет шифрованное сообщение A

6. A
   - собирает сообщения
   - расшифровывает их при помощи лежащего в открытом доступе e_{public}
   - подсчитывает их и публикует результаты

Особенности, преимущества и недостатки

Этот протокол чрезвычайно прост, тем не менее, его достаточно, чтобы защититься от внешнего вмешательства, подделки голосов и дискредитации легитимных избирателей. Однако голосующим приходится абсолютно доверять A, ведь его работа никем не контролируется. С одной стороны, E может предоставить злоумышленнику-покупателю голосов доказательство, как он проголосовал, а с другой – не может проверить, что А правильно учёл или даже получил его бюллетень. Поэтому тривиальный метод применим только в сообществах, где все доверяют друг другу и агентству, отвечающему за подсчёт голосов.

Протокол двух агентств

Он же протокол Нурми – Саломаа – Сантина. Основная идея состоит в том, чтобы заменить одно избирательное агентство двумя, чтобы они контролировали друг друга. Пусть здесь и далее V – регистратор, в обязанности которого входит подготовка списков, а также допуск или недопуск участника до голосования. Последовательность действий выглядит так:

1. V
   - создаёт набор опознавательных меток t_i и утверждает список возможных избирателей
   - отправляет по защищённому каналу по одной метке каждому голосующему
   - отправляет A весь набор меток без информации о том, какая метка кому принадлежит
2. E
   - генерирует $e_{public}, e_{private}$ (для цифровой подписи) и e_{secret} (для того, чтобы ни A, ни посторонний злоумышленник не мог до нужного времени узнать содержимое бюллетеня)
   - e_{public} публикуется
   - формирует сообщение B с выбранным решением
   - подписывает его $e_{private}$
   - прикладывает к нему полученный t_i
   - шифрует при помощи e_{secret}
   - снова прикладывает к шифротексту t_i
   - отправляет шифротекст

     $$\{t_i, encrypt(e_{secret}, \{t_i, sign(e_{private}, B)\})\}$$

     на рассмотрение в A

3. A
    - получает шифротекст. По внешнему тегу оно определяет, что сообщение пришло от легитимного пользователя, но не может определить, ни от какого, ни как он проголосовал
    - выкладывает в открытый доступ полученную пару тег-шифр
4. Опубликованный файл служит сигналом E отправить секретный ключ e_{secret}
5. A
    - собирает ключи
    - производит подсчёт голосов
    - присоединяет к опубликованному шифротексту бюллетень без опознавательного тега, на чём голосование заканчивается.

3 Практическая реализация

3.1 Описание программы

Язык программной реализации – Common Lisp. Основная функция реализует исполнение протокола по шагам, а каждый выполняемый шаг в свою очередь разбит на подшаги. Для удобства представления отображаемые числа представлены в шестнадцатеричном формате, а получившиеся шифротексты сокращены на скриншотах. Используемая в протоколе цифровая подпись: ESIGN. Используемый алгоритм симметричного шифрования: AES-128.

3.2 Результаты тестирования программы

Протестируем работу протокола с числом избирателей, равным 9. Число возможных кандидатов тоже определим равным 9.

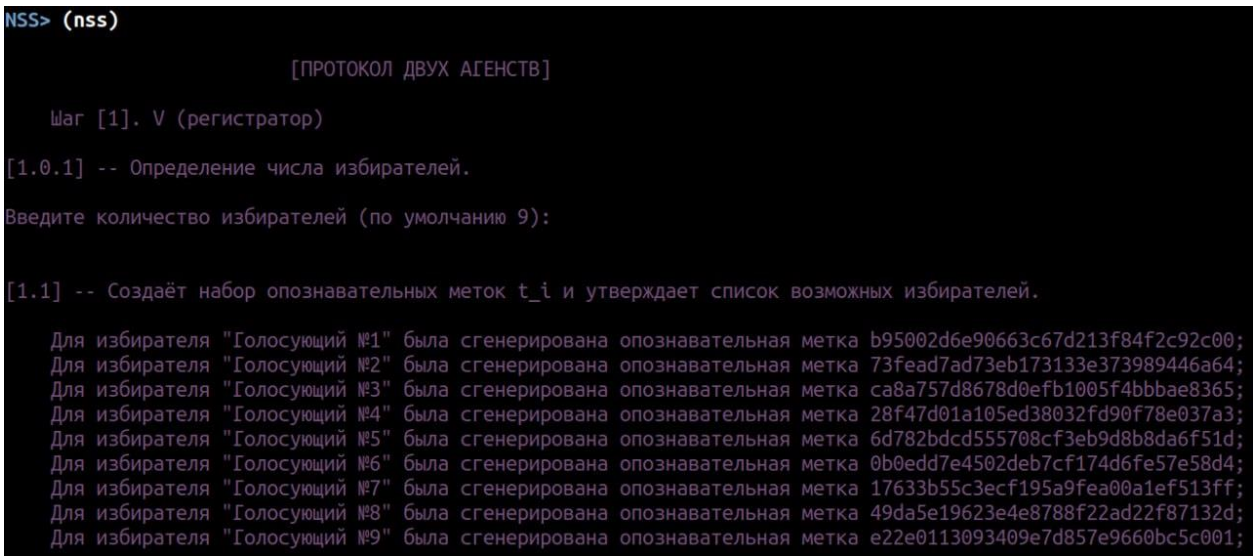

Рисунок 1 – Определение числа избирателей и создание регистратором набора опознавательных меток для них

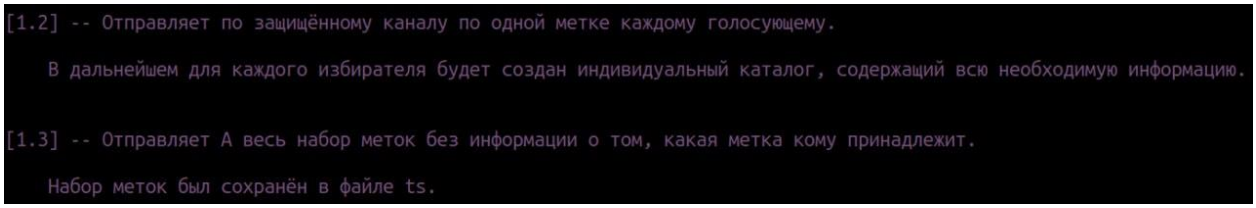

Рисунок 2 – Выполнение подшагов 2 и 3 шага 1 протокола

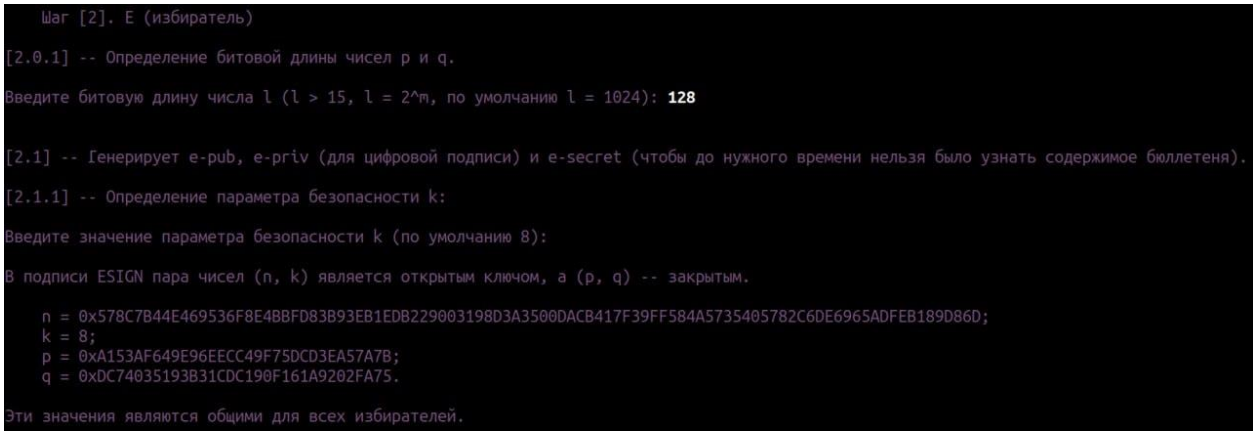

Рисунок 3 – Определение параметров цифровой подписи

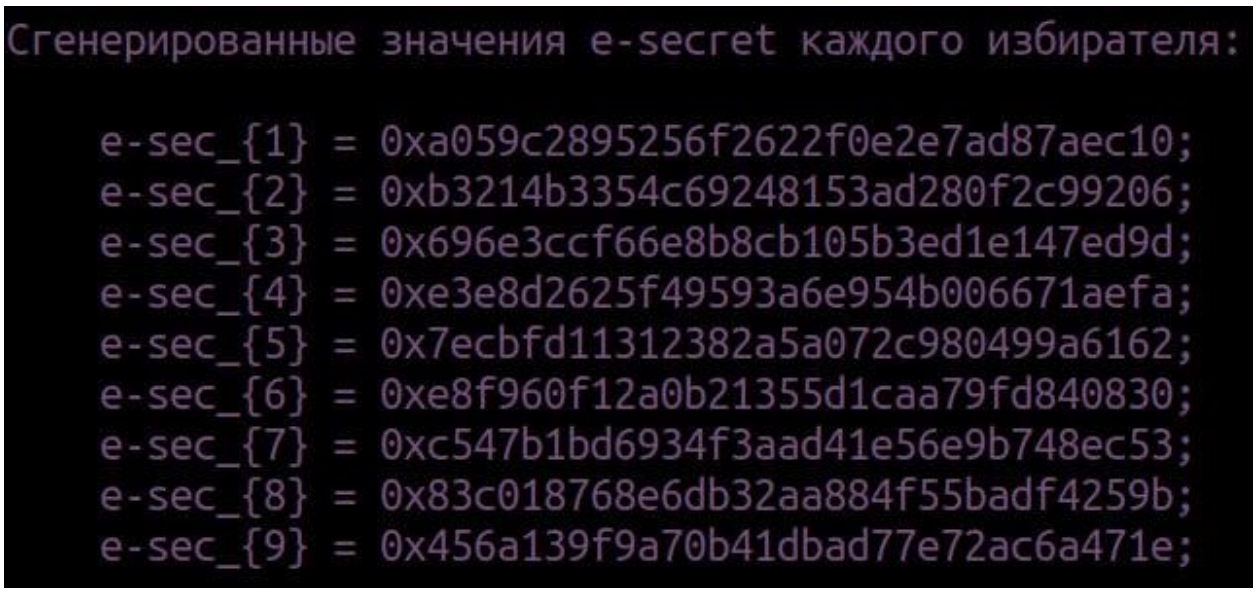

Рисунок 4 – Результат генерации значений e_{secret} для каждого избирателя

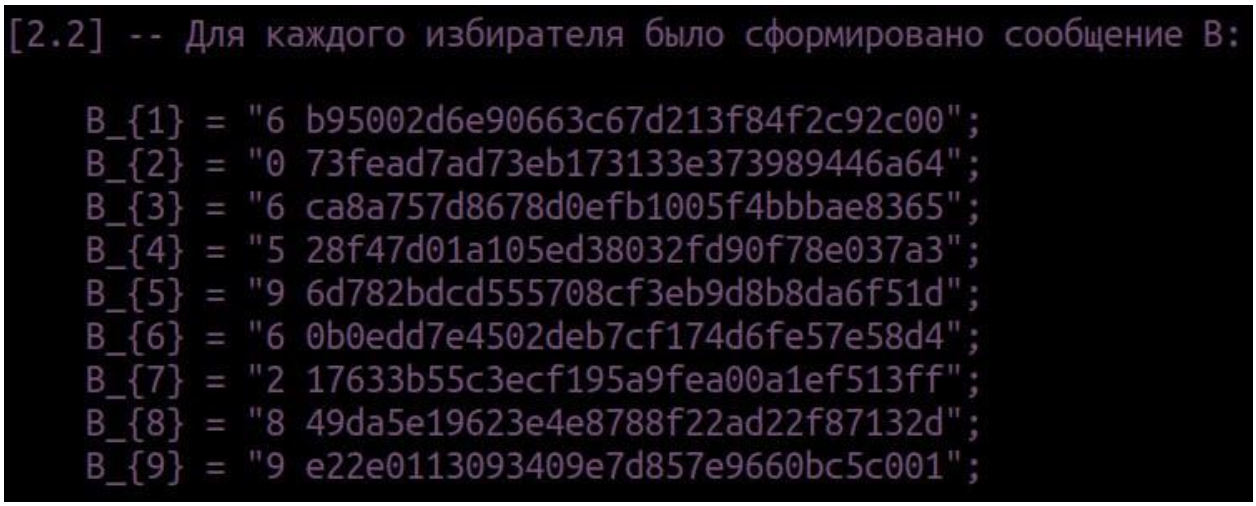

Рисунок 5 – Результат генерации сообщений B для каждого избирателя

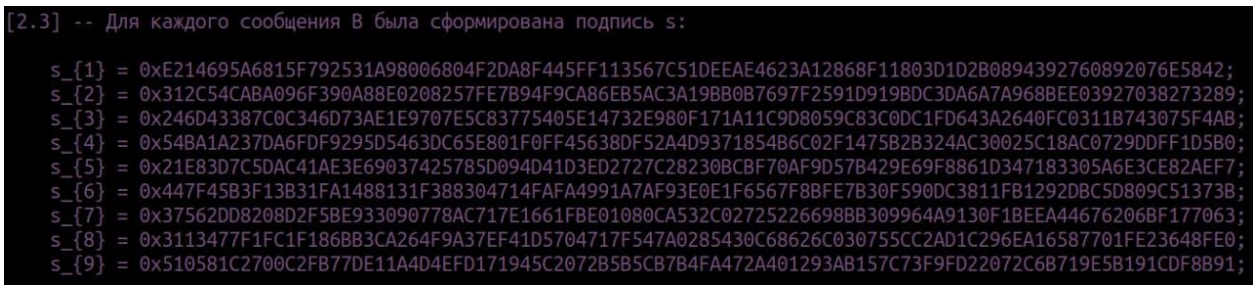

Рисунок 6 – Результат генерации подписи s_i для каждого избирателя

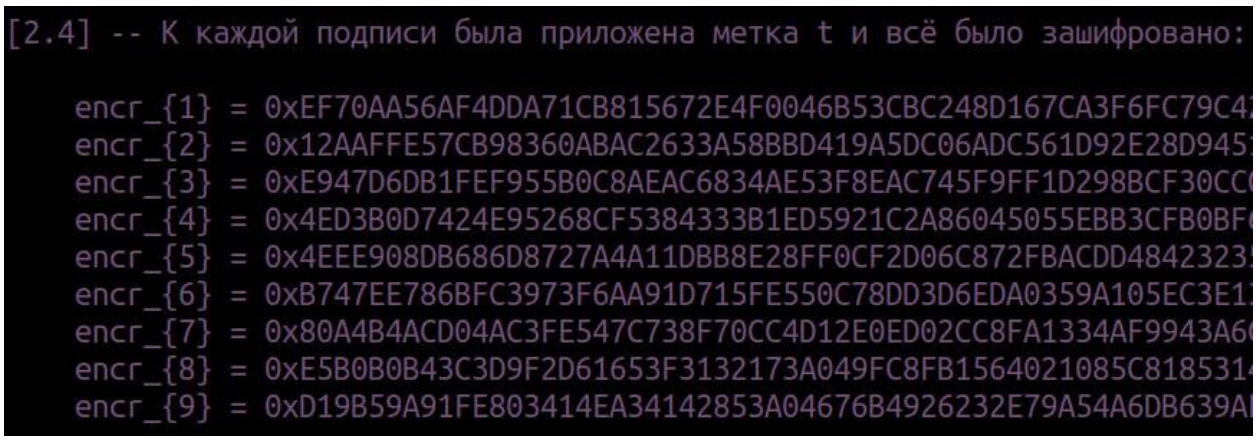

Рисунок 7 – Результат шифрования метки и подписи каждым избирателем

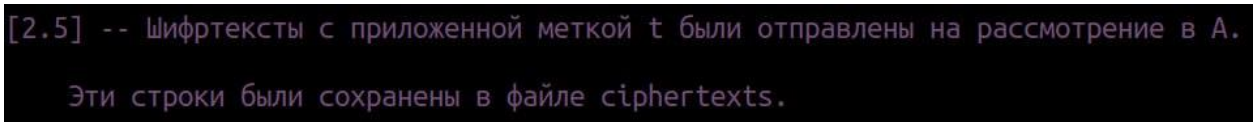

Рисунок 8 – Выполнение подшага 5 шага 2 протокола

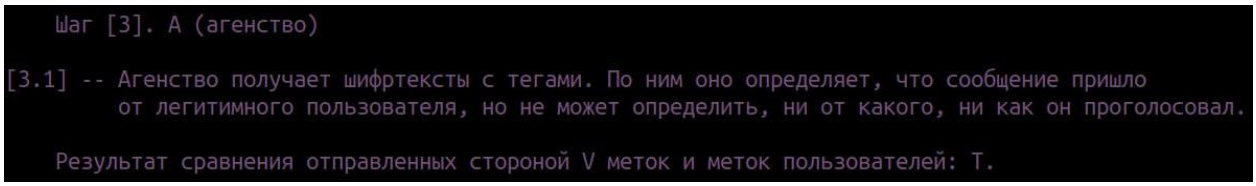

Рисунок 9 – Выполнение подшага 1 шага 3 протокола

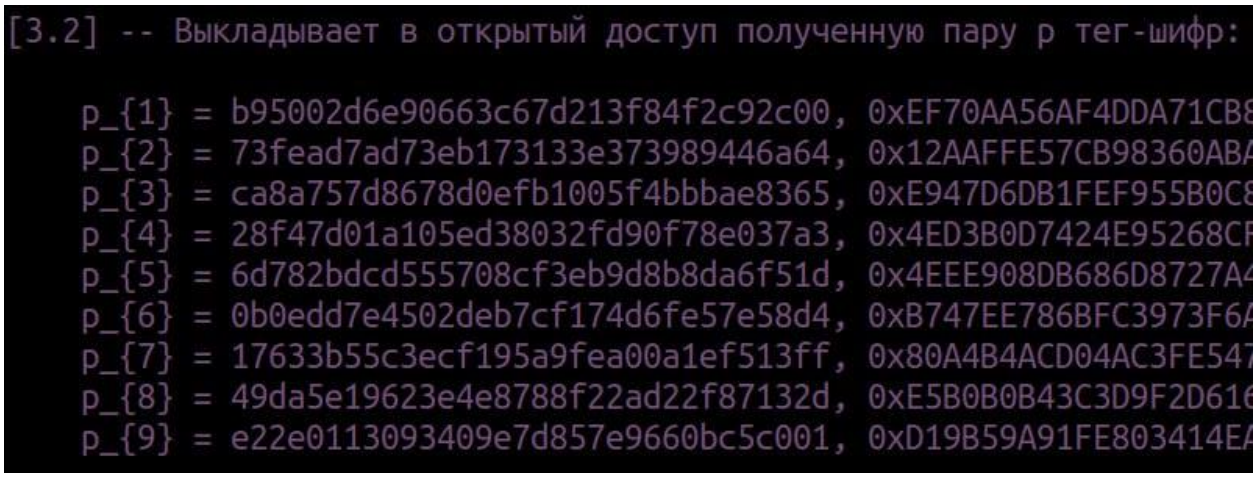

Рисунок 10 – Публикация полученных пар агентством

```
    Шаг [4]. Опубликованный файл служит сигналом E отправить секретный ключ e-secret.


    Шаг [5]. A (агенство)

[5.1] -- Собирает ключи.

    e-sec_{1} = a059c2895256f2622f0e2e7ad87aec10;
    e-sec_{2} = b3214b3354c69248153ad280f2c99206;
    e-sec_{3} = 696e3ccf66e8b8cb105b3ed1e147ed9d;
    e-sec_{4} = e3e8d2625f49593a6e954b006671aefa;
    e-sec_{5} = 7ecbfd11312382a5a072c980499a6162;
    e-sec_{6} = e8f960f12a0b21355d1caa79fd840830;
    e-sec_{7} = c547b1bd6934f3aad41e56e9b748ec53;
    e-sec_{8} = 83c018768e6db32aa884f55badf4259b;
    e-sec_{9} = 456a139f9a70b41dbad77e72ac6a471e;
```

Рисунок 11 – Выполнение шага 4 протокола и сбор агентством ключей

```
[5.2] -- Расшифровывает сообщения.

    p_{1} = b95002d6e90663c67d213f84f2c92c00, 0xE214695A68
    p_{2} = 73fead7ad73eb173133e373989446a64, 0x312C54CABA
    p_{3} = ca8a757d8678d0efb1005f4bbbae8365, 0x246D43387C
    p_{4} = 28f47d01a105ed38032fd90f78e037a3, 0x54BA1A237D
    p_{5} = 6d782bdcd555708cf3eb9d8b8da6f51d, 0x21E83D7C5D
    p_{6} = 0b0edd7e4502deb7cf174d6fe57e58d4, 0x447F45B3F1
    p_{7} = 17633b55c3ecf195a9fea00a1ef513ff, 0x37562DD82C
    p_{8} = 49da5e19623e4e8788f22ad22f87132d, 0x3113477F1F
    p_{9} = e22e0113093409e7d857e9660bc5c001, 0x510581C27C
```

Рисунок 12 – Результат расшифрования агентством полученных шифротекстов

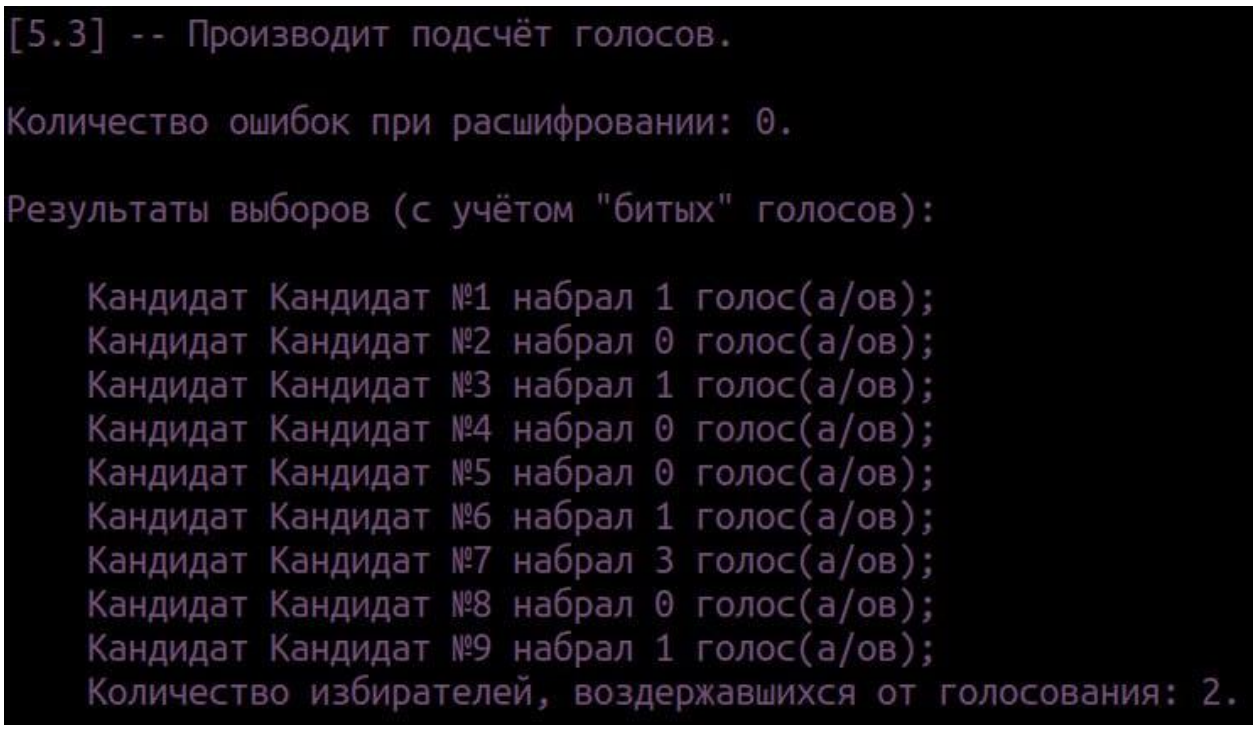

Рисунок 13 – Подсчёт голосов избирателей

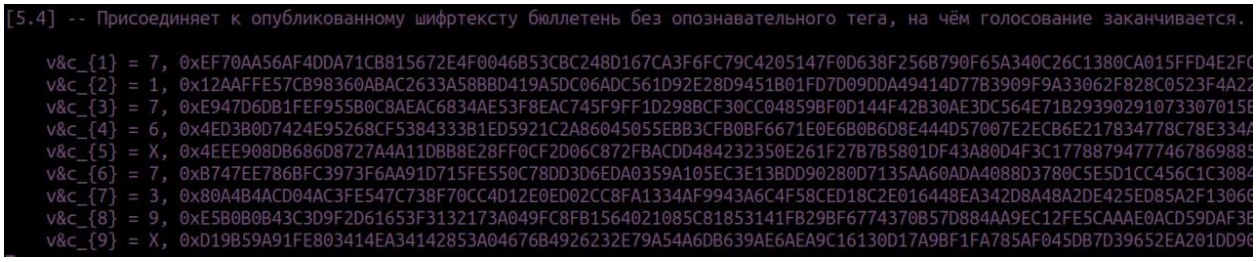

Рисунок 14 – Окончание голосования

Листинг программы

```
(defpackage :crypt
  (:use #:cl)
  (:export #:random-string
           #:aes-encrypt
           #:aes-decrypt
           #:hash))


(in-package :crypt)


(defun random-string (num-bytes)
  (ironclad:byte-array-to-hex-string
   (ironclad:random-data num-bytes)))


(defun ripemd128 (str)
  (ironclad:byte-array-to-hex-string
   (ironclad:digest-sequence
    :ripemd-128
    (ironclad:ascii-string-to-byte-array str))))


(defun get-cipher (key)
  (ironclad:make-cipher :aes
    :mode :ecb
    :key (ironclad:ascii-string-to-byte-array (ripemd128 key))))


(defun aes-encrypt (plaintext key)
  (let ((cipher (get-cipher key))
        (msg (ironclad:ascii-string-to-byte-array plaintext)))
    (ironclad:encrypt-in-place cipher msg)
    (ironclad:octets-to-integer msg)))


(defun aes-decrypt (ciphertext-int key)
  (let ((cipher (get-cipher key))
        (msg (ironclad:integer-to-octets ciphertext-int)))
    (ironclad:decrypt-in-place cipher msg)
    (coerce (mapcar #'code-char (coerce msg 'list)) 'string)))


(defun skein1024 (str)
  (ironclad:byte-array-to-hex-string
   (ironclad:digest-sequence
    :skein1024
```

```
    (ironclad:ascii-string-to-byte-array str))))


(defun hash (str modulo)
  (let ((digest (skein1024 str)))
    (mod (ironclad:octets-to-integer
          (ironclad:ascii-string-to-byte-array digest)) modulo)))
```

```
(defpackage #:aux
  (:use :cl)
  (:export #:write-to-file
           #:read-parse
           #:mod-expt
           #:miller-rabin
           #:ext-gcd
           #:generate-prime))


(in-package #:aux)


(defmacro while (condition &body body)
  `(loop while ,condition
         do (progn ,@body)))


(defun write-to-file (data filename)
  (with-open-file (out filename :direction :output :if-exists :supersede
                                :if-does-not-exist :create)
    (dolist (param data)
      (format out "~a~%" param))))


(defun read-parse (filename &optional (at 0))
  (parse-integer (uiop:read-file-line filename :at at)))


(defun is-pow-of-2? (num)
  (zerop (logand num (1- num))))


(defun mod-expt (base power modulo)
  (setq base (mod base modulo))
  (do ((product 1)) ((zerop power) product)
    (do () ((oddp power))
      (setq base (mod (* base base) modulo)
            power (ash power -1)))
    (setq product (mod (* product base) modulo)
          power (1- power))))


(defun miller-rabin (n &optional (k 10))
  (when (or (= 2 n) (= 3 n)) (return-from miller-rabin t))
  (when (or (< n 2) (= 0 (logand n 1))) (return-from miller-rabin))
  (let* ((n-pred (1- n)) (bound (- n-pred 2)) (t-val n-pred) (s 0) (round 0)
(x))
    (while (= 0 (logand t-val 1)) (setq s (1+ s) t-val (ash t-val -1)))
```

```
    (do () (nil)
      (tagbody next-iteration
         (when (= k round) (return-from miller-rabin t))
         (setq x (mod-expt (+ 2 (random bound)) t-val n))
         (when (or (= 1 x) (= n-pred x))
           (incf round) (go next-iteration))
         (do ((iter 0 (1+ iter))) ((= iter (1- s)) (return-from miller-rabin))
           (setq x (mod (* x x) n))
           (when (= 1 x) (return-from miller-rabin))
           (when (= n-pred x)
             (incf round) (go next-iteration)))))))


(defparameter *base-primes*
  (remove-if-not #'(lambda (prime?) (miller-rabin prime? 12))
                 (loop for prime? from (1+ (ash 1 15)) to (1- (ash 1 16)) by 2
                       collect prime?)))


(defun ext-gcd (a b)
  (let ((s 0) (old-s 1) (r b) (old-r a)
        (quotient) (bezout-t))
    (while (not (zerop r))
      (setq quotient (floor old-r r))
      (psetq old-r r r (- old-r (* quotient r))
             old-s s s (- old-s (* quotient s))))
    (if (zerop b) (setq bezout-t 0)
        (setq bezout-t (floor (- old-r (* old-s a)) b)))
    (list old-r old-s bezout-t)))


(defun generate-even (target-len)
  (apply #'+ (ash 1 (1- target-len))
         (mapcar #'(lambda (bit pow) (* bit (ash 1 pow)))
                 (append (loop for bit from 0 to (- target-len 3)
                               collect (random 2)) '(0))
                 (loop for pow from (- target-len 2) downto 0 collect pow))))


(defun generate-prime (target-len)
  (when (not (is-pow-of-2? target-len))
    (return-from generate-prime))
  (when (= 16 target-len)
    (return-from generate-prime (nth (random (length *base-primes*))
                                     *base-primes*)))
  (let ((prime) (s) (prime?) (req-len (- target-len 16)))
    (tagbody pick-prime
       (setq prime (nth (random (length *base-primes*)) *base-primes*))
       (when (not (miller-rabin prime)) (go pick-prime)))
```

```
(tagbody try-again
   (setq s (generate-even req-len)
         prime? (1+ (* prime s)))
   (if (and (= 1 (mod-expt 2 (1- prime?) prime?))
            (/= 1 (mod-expt 2 s prime?))
            (zerop (logxor (length (write-to-string prime? :base 2))
                           target-len)))
       (return-from generate-prime prime?)
       (go try-again)))))
```

```
(defpackage #:sig
  (:use #:cl)
  (:export #:gen-keys
           #:sign-message
           #:verify-message))


(in-package #:sig)


(defun get-k ()
  (format t "~%Введите значение параметра безопасности k (по умолчанию 8): ")
  (let (k)
    (setq k (read-line))
    (when (or (zerop (length k))
              (null (setq k (parse-integer k :junk-allowed t)))
              (< k 4))
      (setq k 8)) k))


(defun gen-keys (bit-len)
  (let* ((p (aux:generate-prime bit-len))
         (q (aux:generate-prime bit-len))
         (n (* p p q)) (k (get-k)))
    (when (uiop:directory-exists-p "sig")
      (uiop:run-program "rm -r sig"))
    (uiop:run-program "mkdir sig")
    (aux:write-to-file (list p) "sig/p")
    (aux:write-to-file (list q) "sig/q")
    (aux:write-to-file (list n) "sig/n")
    (aux:write-to-file (list k) "sig/k") (list p q n k)))


(defun sign-message (message p q n k)
  (let* ((digest (crypt:hash message n))
         (pq (* p q)) (x (random pq))
         (chi (ceiling (/ (- digest (aux:mod-expt x k n))
                          pq)))
         (h (mod (* chi (cadr (aux:ext-gcd (* k (aux:mod-expt x (1- k) p)) p)))
p)))
    (+ x (* h pq))))


(defun verify-message (message s n k)
  (let ((digest (crypt:hash message n))
        (ksi (ceiling (* 2/3 (integer-length n)))))
    (<= digest (aux:mod-expt s k n) (+ digest (ash 1 ksi)))))
```

```
(defpackage #:ta-aux
  (:use #:cl)
  (:export #:*candidates*
           #:get-n
           #:get-bit-len
           #:gen-voters&ts
           #:form-folders
           #:send-ciphertexts
           #:compare-tags
           #:collect-keys
           #:collect-Bs
           #:decrypt-received
           #:count-voices))


(in-package #:ta-aux)


(defparameter *num-bytes* 16)

(defparameter *candidates* '("Кандидат №1" "Кандидат №2" "Кандидат №3"
                             "Кандидат №4" "Кандидат №5" "Кандидат №6"
                             "Кандидат №7" "Кандидат №8" "Кандидат №9"))


(defun get-n ()
  "Шаг 1. Определение количества избирателей."
  (format t "~%Введите количество избирателей (по умолчанию 9): ")
  (let (n)
    (setq n (read-line))
    (when (or (zerop (length n))
              (null (setq n (parse-integer n :junk-allowed t)))
              (< n 1))
      (setq n 9))
    (aux:write-to-file (list n) "n") n))


(defun get-bit-len ()
  (format t "~%Введите битовую длину числа l (l > 15, l = 2^m, по умолчанию l
= 1024): ")
  (let ((bit-len (read-line)))
    (when (or (zerop (length bit-len))
              (null (setq bit-len (parse-integer bit-len :junk-allowed t)))
              (< bit-len 16)
              (plusp (logand bit-len (1- bit-len))))
      (setq bit-len 1024)) bit-len))


(defun gen-voters&ts ()
```

```
  "Шаг 1. Функция генерирует список избирателей и для каждого избирателя
   опознавательную метку t_i."
  (let* ((n (aux:read-parse "n")) voters ts
         (len (length (write-to-string n))))
    (loop for j from n downto 1
          do (setq voters (cons (format nil "Голосующий №~v,'0d" len j) voters)
                   ts (cons (crypt:random-string *num-bytes*) ts)))
    (aux:write-to-file voters "voters")
    (aux:write-to-file     ts     "ts") (list voters ts)))


(defun gen-secret-keys (n)
  (loop for j from 0 below n
        collect (crypt:random-string *num-bytes*)))


(defun form-votes (n)
  (let ((up-to-choice (1+ (length *candidates*))))
    (loop for j from 0 below n
          collect (random up-to-choice))))


(defun form-messages-to-sign (votes ts)
  (setq votes (mapcar #'write-to-string votes))
  (mapcar #'(lambda (vote t-val)
              (concatenate 'string vote " " t-val)) votes ts))


(defun form-signatures (to-sign p q n k)
  (mapcar #'(lambda (vote)
              (sig:sign-message vote p q n k)) to-sign))


(defun form-ciphertexts (ts sigs keys)
  (let (ciphertexts)
    (setq sigs (mapcar #'write-to-string sigs)
          ciphertexts (mapcar #'(lambda (t-val sig)
                                  (concatenate 'string t-val " " sig))
                              ts sigs)
          ciphertexts (mapcar #'(lambda (plaintext key)
                                  (crypt:aes-encrypt plaintext key))
                              ciphertexts keys))))


(defun form-folders-routine (keys Bs sigs ts ciphertexts n)
  (let ((general-path "voting-data/voter-~a") path)
    (dotimes (j n)
      (setq path (format nil general-path j))
      (uiop:run-program (format nil "mkdir voting-data/voter-~a" j))
```

```
      (aux:write-to-file (list (nth j keys))
                         (concatenate 'string path "/e-secret"))
      (aux:write-to-file (list (nth j Bs))
                         (concatenate 'string path "/B"))
      (aux:write-to-file (list (nth j sigs))
                         (concatenate 'string path "/s"))
      (aux:write-to-file (list (nth j ts))
                         (concatenate 'string path "/t"))
      (aux:write-to-file (list (nth j ciphertexts))
                         (concatenate 'string path "/ciphertext")))))


(defun form-folders (num-users)
  (let* ((p (aux:read-parse "sig/p")) (q (aux:read-parse "sig/q"))
         (n (aux:read-parse "sig/n")) (k (aux:read-parse "sig/k"))
         (secret-keys (gen-secret-keys num-users)) (votes (form-votes num-
users))
         (ts (uiop:read-file-lines "ts"))
         (to-sign (form-messages-to-sign votes ts))
         (sigs (form-signatures to-sign p q n k))
         (ciphertexts (form-ciphertexts ts sigs secret-keys)))
    (when (uiop:directory-exists-p "voting-data")
      (uiop:run-program "rm -r voting-data"))
    (uiop:run-program "mkdir voting-data")
    (form-folders-routine secret-keys to-sign sigs ts ciphertexts num-users)
    (list secret-keys to-sign sigs ciphertexts)))


(defun collect-tag (num-voter)
  (uiop:read-file-line
   (format nil "voting-data/voter-~a/t" num-voter)))


(defun collect-tags (num-users)
  (loop for j from 0 below num-users
        collect (collect-tag j)))


(defun collect-ciphertext (num-voter)
  (uiop:read-file-line
   (format nil "voting-data/voter-~a/ciphertext" num-voter)))


(defun collect-ciphertexts (num-users)
  (loop for j from 0 below num-users
        collect (collect-ciphertext j)))


(defun send-ciphertexts (num-users)
```

```
  (let ((ts (collect-tags num-users))
        (ciphertexts (collect-ciphertexts num-users)))
    (aux:write-to-file (mapcar #'(lambda (t-val ciphertext)
                                   (concatenate 'string t-val " " ciphertext))
                               ts ciphertexts)
                       "ciphertexts") t))


(defun compare-tags ()
  (let (tags received)
    (unless (and (uiop:file-exists-p "ts")
                 (uiop:file-exists-p "ciphertexts"))
      (return-from compare-tags nil))
    (setq tags (uiop:read-file-lines "ts")
          received (uiop:read-file-lines "ciphertexts")
          received (mapcar #'(lambda (ciphertext)
                               (uiop:split-string ciphertext :separator " "))
                           received))
    (null (remove-if #'(lambda (tag)
                         (member tag received :key #'car :test #'equal))
tags))))


(defun collect-key (num-voter)
  (uiop:read-file-line
   (format nil "voting-data/voter-~a/e-secret" num-voter)))


(defun collect-keys (num-users)
  (loop for j from 0 below num-users
        collect (collect-key j)))


(defun decrypt-received (num-users)
  (let ((keys (collect-keys num-users)) ciphertexts)
    (setq ciphertexts (uiop:read-file-lines "ciphertexts")
          ciphertexts (mapcar #'(lambda (ciphertext)
                                  (uiop:split-string ciphertext :separator "
"))
                              ciphertexts)
          ciphertexts (mapcar #'(lambda (ciphertext)
                                  (parse-integer (cadr ciphertext)))
ciphertexts))
    (mapcar #'(lambda (ciphertext key)
                (crypt:aes-decrypt ciphertext key))
            ciphertexts keys)))


(defun collect-B (num-voter)
```

```
  (uiop:read-file-line
   (format nil "voting-data/voter-~a/B" num-voter)))


(defun collect-Bs (num-users)
  (loop for j from 0 below num-users
        collect (collect-B j)))


(defun verify-sigs (num-users)
  (let ((sigs (decrypt-received num-users))
        (Bs (collect-Bs num-users))
        (n (aux:read-parse "sig/n"))
        (k (aux:read-parse "sig/k"))
        (broken 0))
    (setq sigs (mapcar #'(lambda (decrypted)
                           (uiop:split-string decrypted :separator " ")) sigs)
          sigs (mapcar #'(lambda (lst-decrypted)
                           (handler-case (parse-integer (cadr lst-decrypted))
                             (error () (incf broken) t))) sigs))
    (list
     (every #'(lambda (B sig)
                (if sig
                    t
                    (sig:verify-message B sig n k))) Bs sigs) broken)))


(defun make-keyword (num)
  (intern (concatenate 'string ":" (write-to-string num))))


(defun count-voices (num-users)
  (destructuring-bind (correct-sigs? num-broken) (verify-sigs num-users)
    (format t "~2%Количество ошибок при расшифровании: ~d." num-broken)
    (unless correct-sigs?
      (return-from count-voices nil)))
  (let ((voices (collect-Bs num-users))
        (election-results (loop for j from 0 below (* 2 (1+ (length
*candidates*)))
                                if (evenp j)
                                  collect (make-keyword (ash j -1))
                                else collect 0)) voice-keyword)
    (setq voices (mapcar #'(lambda (B)
                             (uiop:split-string B :separator " ")) voices)
          voices (mapcar #'(lambda (B-splitted)
                             (parse-integer (car B-splitted))) voices))
    (dolist (voice voices election-results)
      (setq voice-keyword (make-keyword voice))
      (when (getf election-results voice-keyword)
```

```
(incf (getf election-results voice-keyword))))))
```

```
(defpackage #:nss
  (:use #:cl)
  (:export #:nss))


(in-package #:nss)


(defun stop () (read-line))


(defun step-1-substep-1 ()
  (format t "~2%[1.1] -- Создаёт набор опознавательных меток t_i и утверждает
список возможных избирателей.~2%")
  (destructuring-bind (voters ts) (ta-aux:gen-voters&ts)
    (let ((len (length voters)))
      (do ((j 0 (1+ j))) ((= len j))
        (format t "~4tДля избирателя ~s была сгенерирована опознавательная
метка ~a;~%"
                (nth j voters) (nth j ts))))))


(defun step-1-substep-2 ()
  (format t "~%[1.2] -- Отправляет по защищённому каналу по одной метке каждому
голосующему.~2%")
  (format t "~4tВ дальнейшем для каждого избирателя будет создан индивидуальный
каталог, содержащий всю необходимую информацию.~%"))


(defun step-1-substep-3 ()
  (format t "~%[1.3] -- Отправляет A весь набор меток без информации о том,
какая метка кому принадлежит.~%")
  (format t "~%~4tНабор меток был сохранён в файле ts.~%"))


(defun step-1 ()
  (format t "~%~4tШаг [1]. V (регистратор)~2%[1.0.1] -- Определение числа
избирателей.~%")
  (ta-aux:get-n)
  (step-1-substep-1) (stop)
  (step-1-substep-2) (stop)
  (step-1-substep-3) (stop) t)


(defun step-2-substep-1 (p q n k keys)
  (format t "~%В подписи ESIGN пара чисел (n, k) является открытым ключом, а
(p, q) -- закрытым.~2%")
  (format t "~4tn = 0x~x;~%~4tk = ~a;~%~4tp = 0x~x;~%~4tq = 0x~x.~%"
          n k p q)
```

```
  (format t "~%Эти значения являются общими для всех избирателей.~%")
  (format t "~%Сгенерированные значения e-secret каждого избирателя:~%")
  (let* ((len-keys (length keys)) (len (length (write-to-string len-keys))))
    (dotimes (j len-keys (terpri))
      (format t "~%~4te-sec_{~v,'0d} = 0x~x;" len (1+ j) (nth j keys)))))


(defun step-2-substep-2 (Bs)
  (format t "~%[2.2] -- Для каждого избирателя было сформировано сообщение
B:~%")
  (let* ((len-Bs (length Bs)) (len (length (write-to-string len-Bs))))
    (dotimes (j len-Bs (terpri))
      (format t "~%~4tB_{~v,'0d} = ~s;" len (1+ j) (nth j Bs)))))


(defun step-2-substep-3 (sigs)
  (format t "~%[2.3] -- Для каждого сообщения B была сформирована подпись
s:~%")
  (let* ((len-sigs (length sigs)) (len (length (write-to-string len-sigs))))
    (dotimes (j len-sigs (terpri))
      (format t "~%~4ts_{~v,'0d} = 0x~x;" len (1+ j) (nth j sigs)))))


(defun step-2-substep-4 (ciphers)
  (format t "~%[2.4] -- К каждой подписи была приложена метка t и всё было
зашифровано:~%")
  (let* ((len-ciphers (length ciphers)) (len (length (write-to-string len-
ciphers))))
    (dotimes (j len-ciphers (terpri))
      (format t "~%~4tencr_{~v,'0d} = 0x~x;" len (1+ j) (nth j ciphers)))))


(defun step-2-substep-5 (num-users)
  (ta-aux:send-ciphertexts num-users)
  (format t "~%[2.5] -- Шифртексты с приложенной меткой t были отправлены на
рассмотрение в A.~%")
  (format t "~%~4tЭти строки были сохранены в файле ciphertexts.~%"))


(defun step-2 ()
  (format t "~%~4tШаг [2]. E (избиратель)")
  (format t "~2%[2.0.1] -- Определение битовой длины чисел p и q.~%")
  (let ((bit-len (ta-aux:get-bit-len)) (num-users (aux:read-parse "n")))
    (format t "~2%[2.1] -- Генерирует e-pub, e-priv (для цифровой подписи) и
e-secret (чтобы до нужного времени нельзя было узнать содержимое
бюллетеня).~%")
    (format t "~%[2.1.1] -- Определение параметра безопасности k:~%")
    (destructuring-bind (p q n k) (sig:gen-keys bit-len)
```

```
      (destructuring-bind (keys Bs sigs ciphers) (ta-aux:form-folders num-
users)
        (step-2-substep-1 p q n k keys) (stop)
        (step-2-substep-2 Bs          ) (stop)
        (step-2-substep-3 sigs        ) (stop)
        (step-2-substep-4 ciphers     ) (stop)
        (step-2-substep-5 num-users   ) (stop) t))))


(defun step-3-substep-1 ()
  (format t "~2%[3.1] -- Агенство получает шифртексты с тегами. По ним оно
определяет, что сообщение пришло
         от легитимного пользователя, но не может определить, ни от какого, ни
как он проголосовал.")
  (let ((correct-tags? (ta-aux:compare-tags)))
    (format t "~2%~4tРезультат сравнения отправленных стороной V меток и меток
пользователей: ~a.~%"
            correct-tags?) correct-tags?))


(defun step-3-substep-2 ()
  (format t "~%[3.2] -- Выкладывает в открытый доступ полученную пару p тег-
шифр:~%")
  (let* ((ciphertexts (uiop:read-file-lines "ciphertexts"))
         (len-ciph (length ciphertexts))
         (len (length (write-to-string len-ciph))) pair)
    (setq ciphertexts (mapcar #'(lambda (ciphertext)
                                  (uiop:split-string ciphertext :separator "
"))
                              ciphertexts))
    (dotimes (j len-ciph (terpri))
      (setq pair (nth j ciphertexts))
      (format t "~%~4tp_{~v,'0d} = ~a, 0x~x;"
              len (1+ j) (car pair) (parse-integer (cadr pair))))))


(defun step-3 ()
  (format t "~%~4tШаг [3]. A (агенство)")
  (when (not (step-3-substep-1))
    (format t "~%Не все значения меток совпали! Завершение протокола.")
    (return-from step-3 nil)) (stop)
  (step-3-substep-2) (stop) t)


(defun step-4 ()
  (format t "~%~4tШаг [4]. Опубликованный файл служит сигналом E отправить
секретный ключ e-secret.~%")
  (stop) t)
```

```
(defun step-5-substep-1 (num-users)
  (format t "~2%[5.1] -- Собирает ключи.~%")
  (let* ((keys (ta-aux:collect-keys num-users))
         (len-keys (length keys))
         (len (length (write-to-string len-keys))))
    (dotimes (j len-keys (terpri))
      (format t "~%~4te-sec_{~v,'0d} = ~a;" len (1+ j) (nth j keys)))))


(defun step-5-substep-2 (num-users)
  (format t "~%[5.2] -- Расшифровывает сообщения.~%")
  (let* ((decrypted (ta-aux:decrypt-received num-users))
         (len-decr (length decrypted))
         (len (length (write-to-string len-decr))) pair)
    (setq decrypted (mapcar #'(lambda (cipher)
                                (uiop:split-string cipher :separator " "))
                            decrypted))
    (dotimes (j len-decr (terpri))
      (setq pair (nth j decrypted))
      (format t "~%~4tp_{~v,'0d} = ~a, 0x~x;"
              len (1+ j) (car pair) (handler-case (parse-integer (cadr pair))
                                      (error () (cadr pair)))))))


(defun step-5-substep-3 (num-users)
  (format t "~%[5.3] -- Производит подсчёт голосов.")
  (let* ((election-results (ta-aux:count-voices num-users))
         (list-voices (loop for j from 0 below (length election-results)
                            when (oddp j)
                              collect (nth j election-results))))
    (format t "~2%Результаты выборов (с учётом \"битых\" голосов):~%")
    (dotimes (j (length ta-aux:*candidates*) (terpri))
      (format t "~%~4tКандидат ~a набрал ~d голос(а/ов);"
              (nth j ta-aux:*candidates*) (nth j list-voices)))
    (format t "~4tКоличество избирателей, воздержавшихся от голосования: ~d.~%"
            (car (last list-voices)))))


(defun step-5-substep-4 (num-users)
  (format t "~%[5.4] -- Присоединяет к опубликованному шифртексту бюллетень
без опознавательного тега, на чём голосование заканчивается.~%")
  (let* ((votes (ta-aux:collect-Bs num-users))
         (ciphers (uiop:read-file-lines "ciphertexts"))
         (len (length (write-to-string num-users)))
         (num-cands (length ta-aux:*candidates*)) vote)
    (setq votes (mapcar #'(lambda (vote&tag)
                            (car (uiop:split-string vote&tag :separator " ")))
votes)
```

```
          votes (mapcar #'parse-integer votes)
          ciphers (mapcar #'(lambda (tag&cipher)
                              (cadr (uiop:split-string tag&cipher :separator "
"))) ciphers)
          ciphers (mapcar #'parse-integer ciphers))
    (dotimes (j num-users (terpri))
      (setq vote (1+ (nth j votes)))
      (when (= (1+ num-cands) vote)
        (setq vote 'X))
      (format t "~%~4tv&c_{~v,'0d} = ~a, 0x~x;"
              len (1+ j) vote (nth j ciphers)))))


(defun step-5 ()
  (let ((num-users (aux:read-parse "n")))
    (format t "~%~4tШаг [5]. A (агенство)")
    (step-5-substep-1 num-users) (stop)
    (step-5-substep-2 num-users) (stop)
    (step-5-substep-3 num-users) (stop)
    (step-5-substep-4 num-users) t))


(defun nss ()
  (ignore-errors (uiop:run-program "make"))
  (format t "~%~25t[ПРОТОКОЛ ДВУХ АГЕНСТВ]~%")
  (step-1) (step-2)
  (step-3) (step-4)
  (step-5) t)
```