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

Разделение секрета

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 схему разделения секрета Блэкли.

2 Теоретические сведения

Разделяемым секретом в схеме Блэкли является одна из координат точки в m-мерном пространстве. Долями секрета, раздаваемые сторонам, являются уравнения $(m-1)$-мерных гиперплоскостей. Для восстановления точки необходимо знать m уравнений гиперплоскостей. Менее, чем m сторон не смогут восстановить секрет, так как множеством пересечения $m-1$ плоскостей является прямая, и секрет не может быть восстановлен.

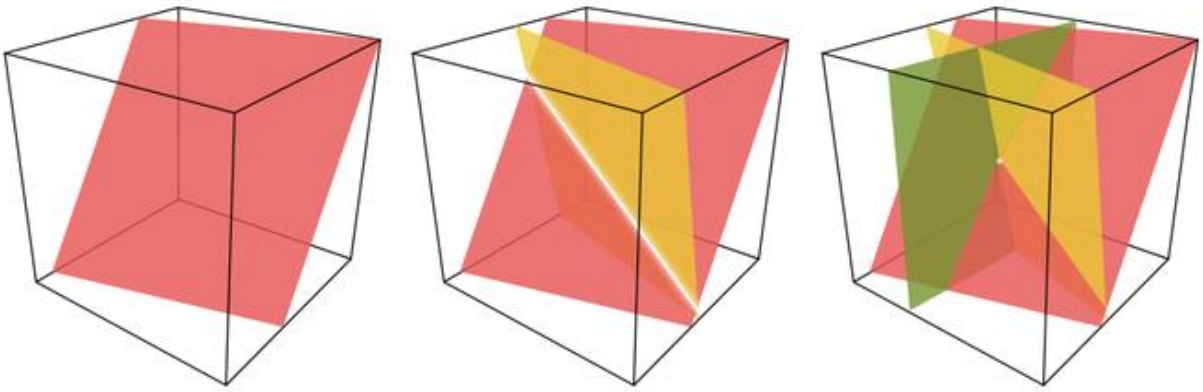

Рисунок 1 – Пример схемы Блэкли в трёх измерениях

Генерация точки

Пусть нужно реализовать (k, n)-пороговую схему, то есть секрет M разделить между n сторонами так, чтобы любые k из них могли восстановить секрет. Для этого выбирается большое простое число $p > M$, по модулю которого будет строиться поле $GF(p)$. Случайным образом выбираются числа $b_2, \dots, b_k \in GF(p)$. Тем самым задается точка $(M, b_2, \dots, b_k)$ в k-мерном пространстве, первая координата которой является секретом.

Раздача секрета

Для каждой стороны $P_i, i = (1, \dots, n)$ случайным образом выбираются коэффициенты $a_{1i}, a_{2i}, \dots, a_{ki}$, равномерно распределённые в поле $GF(p)$. Так как

уравнение плоскости имеет вид $a_{1i} \cdot x_1 + a_{2i} \cdot x_2 + \cdots + a_{ki} \cdot x_k + d_i = 0$, для каждой стороны необходимо вычислить коэффициенты d_i:

$$d_1 = -(a_{11} \cdot M + a_{21} \cdot b_2 + \cdots + a_{k1} \cdot b_k) \; mod \; p,$$

$$d_2 = -(a_{12} \cdot M + a_{22} \cdot b_2 + \cdots + a_{k2} \cdot b_k) \; mod \; p,$$

...

$$d_i = -(a_{1i} \cdot M + a_{2i} \cdot b_2 + \cdots + a_{ki} \cdot b_k) \; mod \; p,$$

...

$$d_n = -(a_{1n} \cdot M + a_{2n} \cdot b_2 + \cdots + a_{kn} \cdot b_k) \; mod \; p.$$

При этом необходимо следить, чтобы любые k уравнений были линейно независимы. В качестве долей секрета сторонам раздают набор коэффициентов, задающих уравнение гиперплоскости.

Восстановление секрета

Для восстановления секрета любым k сторонам необходимо собраться вместе и из имеющихся долей секрета составить уравнения для отыскания точки пересечения гиперплоскостей:

$$\begin{cases} (a_{11} \cdot x_1 + a_{21} \cdot x_2 + \cdots + a_{k1} \cdot x_k + d_1) \; mod \; p = 0, \\ (a_{12} \cdot x_1 + a_{22} \cdot x_2 + \cdots + a_{k2} \cdot x_k + d_2) \; mod \; p = 0, \\ \qquad\qquad\qquad\qquad ... \\ (a_{1k} \cdot x_1 + a_{2k} \cdot x_2 + \cdots + a_{kk} \cdot x_k + d_k) \; mod \; p = 0. \end{cases}$$

Решение системы даёт точку в k-мерном пространстве, первая координата которой и есть разделяемый секрет. Систему можно решать любым известным способом, например, методом Гаусса, но при этом необходимо проводить вычисления в поле $GF(p)$.

Если число участников встречи будет меньше, чем k, например, $k - 1$, то результатом решения системы уравнений, составленной из имеющегося набора коэффициентов, будет прямая в k-мерном пространстве. Тем самым множество допустимых значений секрета, удовлетворяющих полученной системе, в

точности совпадает с полным числом элементов поля $GF(p)$, и секрет равновероятно может принимать любое значение из этого поля. Таким образом, участники, собравшись вместе, не получат никакой новой информации о разделённом секрете.

3 Практическая реализация

3.1 Описание программы

Язык программной реализации – Common Lisp. Программа реализует три основные функции протокола – генерацию точки, разделение секрета и восстановление секрета. Каждая из функций осуществляет своё выполнение по шагам согласно соответствующему пункту схемы. Для удобства отображения чисел используется шестнадцатеричная система счисления. Для проверки получившейся СЛУ на k-линейную независимость при раздаче секрета и решения получившейся СЛУ при восстановлении секрета используется метод Гаусса решения СЛУ в конечных полях.

3.2 Результаты тестирования программы

Рассмотрим результат работы программы при введённых данных $n = 6$ и $k = 3$ (т.е. секрет разделяется между 6-ю сторонами так, что любые 3 могут восстановить секрет), $M = 99991$.

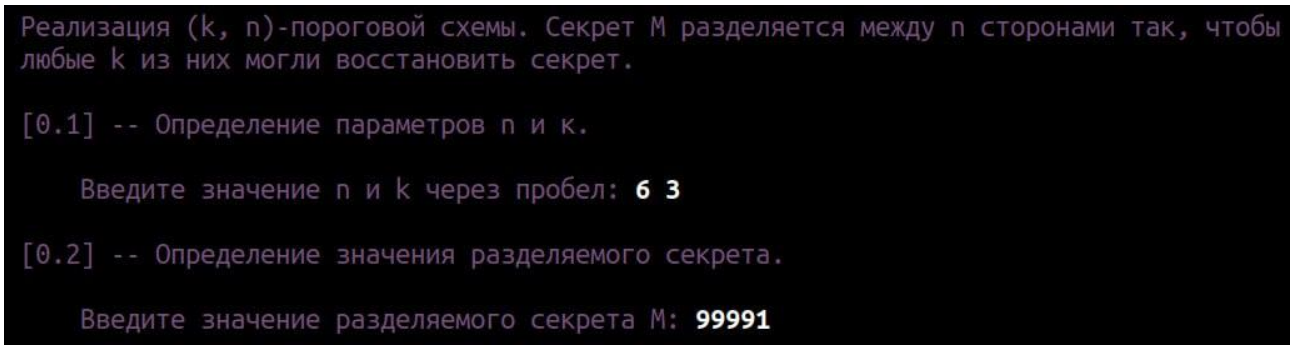

Рисунок 1 – Определение рабочих параметров на этапе генерации точки

```
[0.3] -- Генерация простого числа p.

    Битовая длина M: 17.

    Введите битовую длину l числа p (l = 2^s, l > 15, l > битовая длина(M)): 64

    Было сгенерировано число p: 0x80C065EBAAD9C143.
```

Рисунок 2 – Определение характеристики поля на этапе генерации точки

```
[0.4] -- Случайным образом выбираются числа b_2, ..., b_k in GF(p). Таким образом задаётся точка
         (M, b_2, ..., b_k) в k-мерном пространстве, первая координата которой является секретом.

    Были выбраны координаты:

        b_ 2 = 0x63709CD3868EC7AF;
        b_ 3 = 0x30CD408FE2216E61;
```

Рисунок 3 – Получившиеся координаты сгенерированной точки

```
BLAKLEY> (share-secret)

[1.1] -- Для каждой стороны P_i, i = (1, ..., n) случайным образом выбираются коэффициенты
         a_{1i}, a_{2i}, ..., 2_{ki}, равномерно распределённые в GF(p). Также для каждой
         стороны необходимо вычислить коэффициенты d_i = -(a_{1i} * M + a_{2i} * b_2 + ...
         + a_{ki} * b_k) (mod p)

    Для стороны P_ 1 были выбраны коэффициенты:

        a_{ 1 1} = 0x6C26E97EAFB03831;
        a_{ 2 1} = 0x4F44B98E9BF84366;
        a_{ 3 1} = 0x10FBC40C2BDB3D3F;
        d_ 1     = 0x50276D57DE669187;
```

Рисунок 4 – Определение коэффициентов уравнения для стороны P_1 на этапе разделения секрета

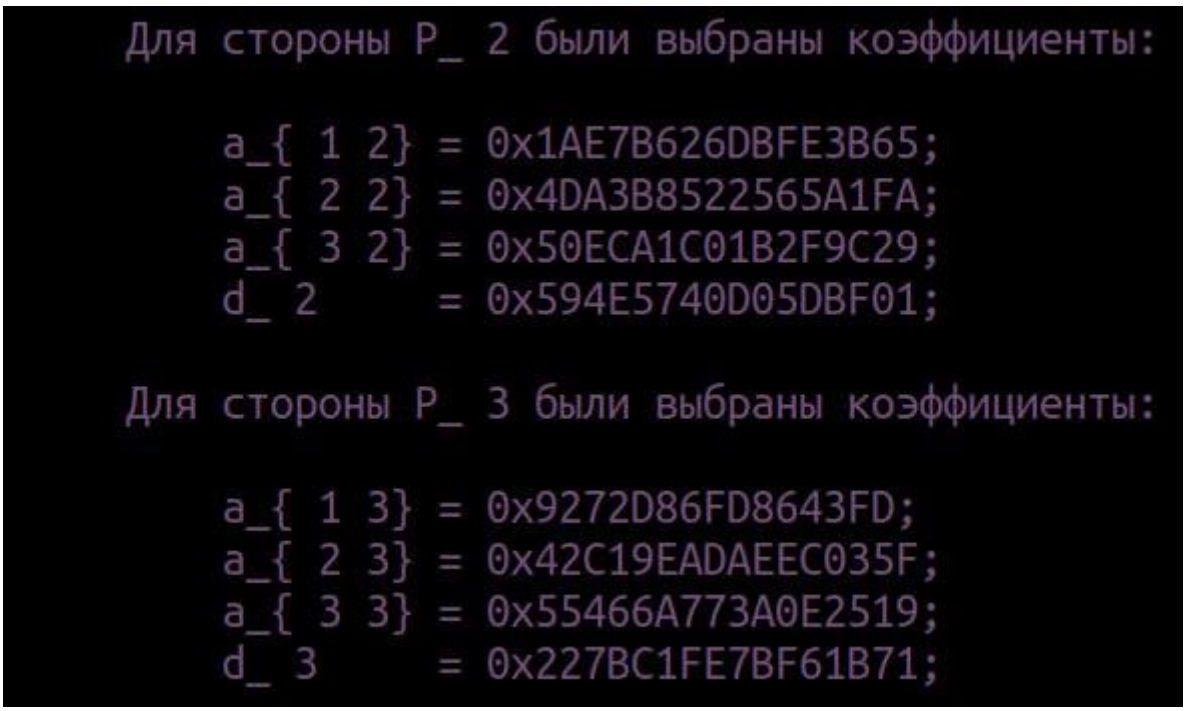

Рисунок 5 – Определение коэффициентов уравнения для сторон P_2, P_3 на этапе разделения секрета

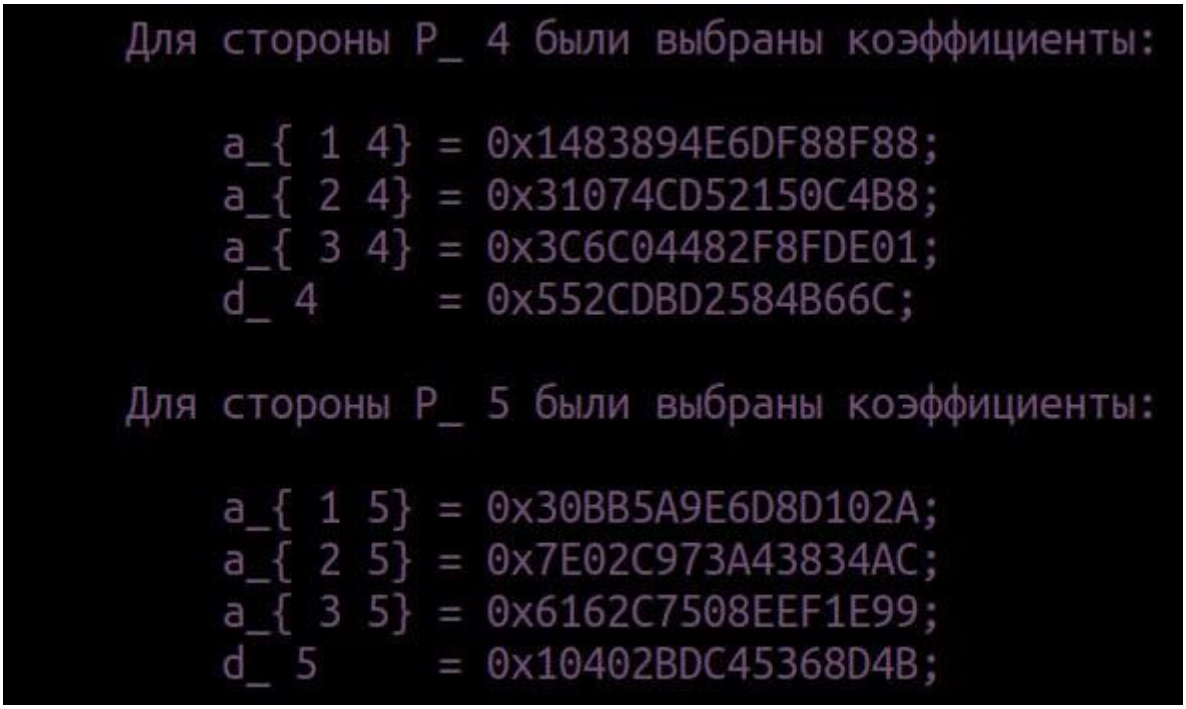

Рисунок 6 – Определение коэффициентов уравнения для сторон P_4, P_5 на этапе разделения секрета

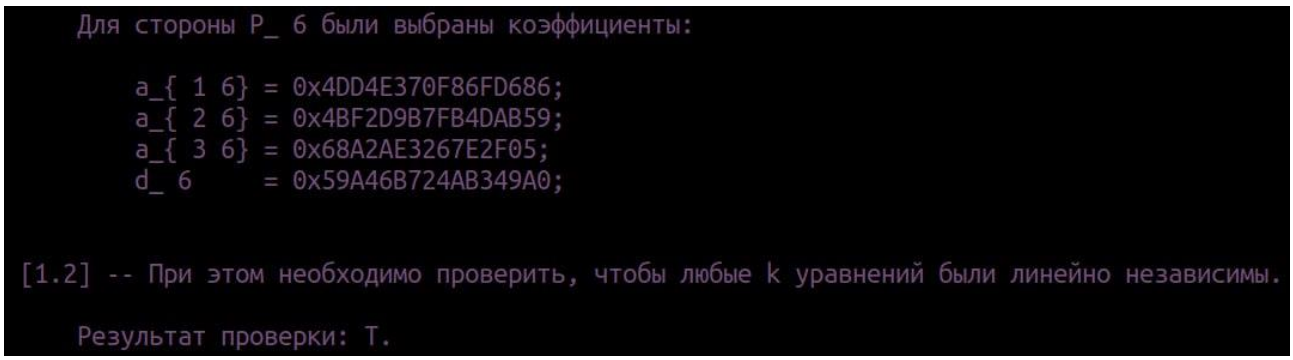

Рисунок 7 – Определение коэффициентов уравнения для стороны P_6 на этапе разделения секрета и проверки ЛНЗ получившейся СЛУ

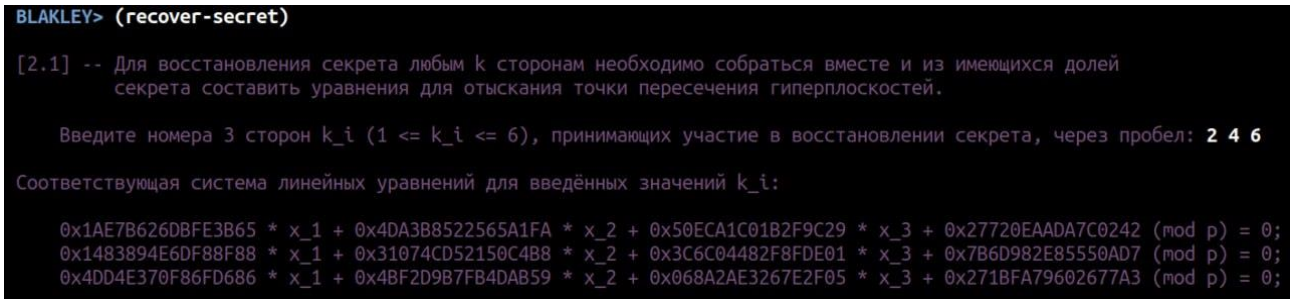

Рисунок 8 – Определение сторон, восстанавливающих секрет

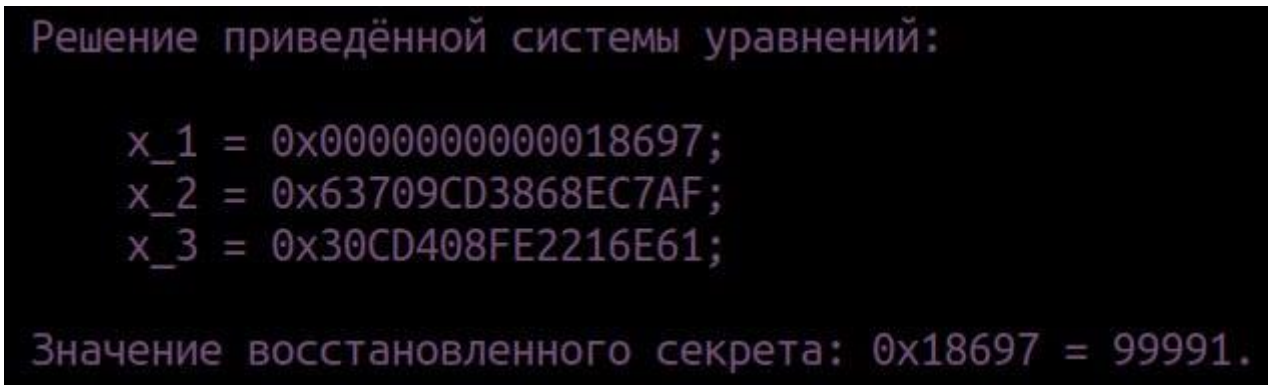

Рисунок 9 – Решение полученной СЛУ и восстановление секрета

Листинг программы

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


(in-package #:aux)


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


(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 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 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 20)) (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 #:gauss
  (:use #:cl)
  (:export #:row-echelon
           #:reduced-row-echelon
           #:row-dimension
           #:column-dimension
           #:switch-rows
           #:multiply-row
           #:add-row))

(in-package #:gauss)


(declaim (inline row-dimension column-dimension))


(defun row-dimension (a)
  "Возвращает количество строк матрицы A."
  (array-dimension a 0))


(defun column-dimension (a)
  "Возвращает количество столбцов матрицы A."
  (array-dimension a 1))


(defun switch-rows (a i j)
  "Деструктивно меняет строки i и j матрицы A, возвращает A."
  (dotimes (k (column-dimension a) a)
    (psetf (aref a i k) (aref a j k)
           (aref a j k) (aref a i k))))


(defun multiply-row (a i alpha p)
  "Деструктивно умножает i-ую строку матрицы A на alpha по модулю p, возвращает
A."
  (dotimes (k (column-dimension a) a)
    (setf (aref a i k) (mod (* (aref a i k) alpha) p))))


(defun add-row (a i j alpha p)
  "Деструктивно добавить к i-ой строке матрицы A её j-ую строку, умноженную на
alpha,
   по модулю p, возвращает A."
  (dotimes (k (column-dimension a) a)
    (setf (aref a i k) (mod (+ (aref a i k) (* (aref a j k) alpha)) p))))


(defun eliminate-column-below (a i j p)
```

```
  "Предполагая, что a[i, j] не равен нулю, деструктивно обнуляет ненулевые
коэффициенты
   под a[i, j]. Возвращает A."
  (loop with inv = (cadr (aux:ext-gcd (aref a i j) p))
        for k from (+ i 1) below (array-dimension a 0)
        do (add-row a k i (- (* (aref a k j) inv)) p)
        finally (return a)))


(defun eliminate-column-above (a i j p)
  "Предполагая, что a[i, j] не равен нулю, деструктивно обнуляет ненулевые
коэффициенты
   над a[i, j]. Возвращает A."
  (loop with inv = (cadr (aux:ext-gcd (aref a i j) p))
        for k below i
        do (add-row a k i (- (* (aref a k j) inv)) p)
        finally (return a)))


(defun find-pivot-row (a i j p)
  "Возвращает первый встреченный индекс строки, начиная с i, имеющей ненулевой
коэффициент
   в j-ом столбце, или nil, в случае, если такой индекс не найден."
  (loop for k from i below (row-dimension a)
        unless (zerop (mod (aref a k j) p))
          do (return k)
        finally (return nil)))


(defun find-pivot-column (a i j p)
   "Возвращает первый встреченный индекс столбца, начиная с i, имеющей ненулевой
коэффициент
   в j-ой строке, или nil, в случае, если такой индекс не найден."
  (loop for k from j below (column-dimension a)
        unless (zerop (mod (aref a i k) p))
          do (return k)
        finally (return nil)))


(defun row-echelon (a p)
  "Приведение матрицы A к ступенчатому виду."
  (loop with row-dimension = (row-dimension a)
        with column-dimension = (column-dimension a)
        with current-row = 0
        with current-col = 0
        while (and (< current-row row-dimension)
                   (< current-col column-dimension))
        for pivot-row = (find-pivot-row a current-row current-col p)
        do (when pivot-row
```

```
            (unless (= pivot-row current-row)
              (switch-rows a pivot-row current-row))
            (eliminate-column-below a current-row current-col p)
            (incf current-row))
        do (incf current-col)
        finally (return a)))


(defun reduce-row-echelon (a p)
  "Функция выполняет обратный ход метода Гаусса в предположении, что матрица A
   уже в ступенчатом виде."
  (loop for i below (row-dimension a)
        for j = (find-pivot-column a i 0 p) then (find-pivot-column a i j p)
        while j
        unless (= 1 (aref a i j))
          do (multiply-row a i (cadr (aux:ext-gcd (aref a i j) p)) p)
        do (eliminate-column-above a i j p)
        finally (return a)))


(defun reduced-row-echelon (a p)
  "Функция возвращает приведённую ступенчатую форму матрицы A."
  (reduce-row-echelon (row-echelon a p) p))
```

```
(defpackage #:bl-aux
  (:use :cl)
  (:export #:get-n-k   #:get-M
           #:gen-p     #:gen-coords
           #:gen-cfs   #:k-linear-independent?
           #:get-ks    #:recover-secret
           #:zerop-row #:array-slice))


(in-package #:bl-aux)


(defun get-n-k ()
  (format t "~%[0.1] -- Определение параметров n и к.~%")
  (format t "~%~4tВведите значение n и k через пробел: ")
  (let ((n-k))
    (tagbody try-again
       (setq n-k (uiop:split-string (read-line) :separator " ")
             n-k (mapcar #'(lambda (int?) (parse-integer int? :junk-allowed t))
n-k))
       (destructuring-bind (n k) n-k
         (when (or (null n) (null k) (< k 2) (< n k))
           (format t "~%~4tНекорректное значение параметров n и / или k!
Попробуйте ввести их снова: ")
           (go try-again))
         (aux:write-to-file (list n) "n")
         (aux:write-to-file (list k) "k"))) n-k))


(defun get-M ()
  (format t "~%[0.2] -- Определение значения разделяемого секрета.~%")
  (format t "~%~4tВведите значение разделяемого секрета M: ")
  (let ((M))
    (tagbody try-again
       (setq M (parse-integer (read-line) :junk-allowed t))
       (when (or (null M) (< M 0))
         (format t "~%~4tНекорректное значение M! Попробуйте ввести его снова:
")
         (go try-again)))
    (aux:write-to-file (list M) "M") M))


(defun gen-p ()
  (format t "~%[0.3] -- Генерация простого числа p.~%")
  (let* ((M (aux:read-parse "M"))
         (bit-len-M (length (write-to-string M :base 2)))
         (bit-len-p) (p))
    (format t "~%~4tБитовая длина M: ~a.~%" bit-len-M)
```

```
    (format t "~%~4tВведите битовую длину l числа p (l = 2^s, l > 15, l > битовая
длина(M)): ")
    (tagbody try-again
       (setq bit-len-p (parse-integer (read-line) :junk-allowed t))
       (when (or (null bit-len-p) (>= bit-len-M bit-len-p)
                 (null (setq p (aux:generate-prime bit-len-p))))
         (format t "~%~4tНекорректное значение битовой длины l! Попробуйте ввести
l снова: ")
         (go try-again)))
    (format t "~%~4tБыло сгенерировано число p: 0x~x.~%" p)
    (aux:write-to-file (list p) "p") p))


(defun gen-coords ()
  (format t "~%[0.4] -- Случайным образом выбираются числа b_2, ..., b_k \in
GF(p). Таким образом задаётся точка
         (M, b_2, ..., b_k) в k-мерном пространстве, первая координата которой
является секретом.")
  (format t "~2%~4tБыли выбраны координаты:~%")
  (let ((k (aux:read-parse "k")) (p (aux:read-parse "p")) (coords))
    (do ((j 2 (1+ j))) ((> j k) (setq coords (reverse coords)))
      (setq coords (cons (random p) coords)))
    (do ((j 2 (1+ j))) ((> j k))
      (format t "~%~8tb_~2d = 0x~x;" j (nth (- j 2) coords)))
    (aux:write-to-file coords "coords")))


(defun gen-cfs ()
  (format t "~%[1.1] -- Для каждой стороны P_i, i = (1, ..., n) случайным образом
выбираются коэффициенты
         a_{1i}, a_{2i}, ..., 2_{ki}, равномерно распределённые в GF(p). Также
для каждой
         стороны необходимо вычислить коэффициенты d_i = -(a_{1i} * M + a_{2i} *
b_2 + ...
         + a_{ki} * b_k) (mod p)")
  (let* ((n (aux:read-parse "n")) (k (aux:read-parse "k"))
         (p (aux:read-parse "p")) (M (aux:read-parse "M"))
         (coords   (cons   M   (mapcar   #'parse-integer   (uiop:read-file-lines
"coords"))))
         (cur-cfs) (cfs) (d) (ds)) (terpri)
    (do ((j 0 (1+ j))) ((= n j) (setq cfs (reverse cfs) ds (reverse ds)))
      (setq cur-cfs (loop for i from 1 to k collect (random p)))
      (format t "~%~4tДля стороны P_~2d были выбраны коэффициенты: ~2%" (1+ j))
      (do ((s 0 (1+ s))) ((= k s))
        (format t "~8ta_{~2d~2d} = 0x~x;~%" (1+ s) (1+ j) (nth s cur-cfs)))
      (setq cfs (cons cur-cfs cfs)
            d (mod (- (reduce #'+ (mapcar #'* cur-cfs coords))) p)
            ds (cons d ds))
      (format t "~8td_~2d     = 0x~x;~%" (1+ j) d))
```

```
    (aux:write-to-file cfs "cfs") (aux:write-to-file  ds  "ds")))


(defun load-cfs (filename)
  (let ((cfs))
    (with-open-file (in filename)
      (with-standard-io-syntax
        (setq cfs (mapcar #'read-from-string
                          (uiop:read-file-lines in)))))))


(defun array-slice (arr row)
  "Возвращает строку с номером row из двумерного массива arr."
  (let ((arr-dim (gauss:column-dimension arr)))
    (make-array arr-dim
                :displaced-to arr
                :displaced-index-offset (* row arr-dim))))


(defun zerop-row (arr row)
  (every #'zerop (array-slice arr row)))


(defun k-linear-independent? ()
  (format t "~%[1.2] -- При этом необходимо проверить, чтобы любые k уравнений
были линейно независимы.")
  (let* ((cfs (load-cfs "cfs")) (p (aux:read-parse "p")) (k (aux:read-parse "k"))
         (ds (mapcar #'parse-integer (uiop:read-file-lines "ds")))
         (equations (mapcar #'(lambda (cfsi di) (append cfsi (list di))) cfs ds))
         (gauss-solution  (gauss:reduced-row-echelon  (make-array  (list  (length
equations)
                                                                          (length
(car equations)))
                                                                 :initial-
contents equations) p))
         (k-linear-independent?))

    (setq k-linear-independent?
          (= k (length (remove-if #'(lambda (row-index)
                                      (bl-aux:zerop-row   gauss-solution   row-
index))
                                  (loop for idx from 0 to k collect idx)))))
    (format t "~2%~4tРезультат проверки: ~a.~%" k-linear-independent?)
    k-linear-independent?))


(defun get-ks ()
  (let ((n (aux:read-parse "n")) (k (aux:read-parse "k")) (ks))
```

```
    (format t "~2%~4tВведите номера ~a сторон k_i (1 <= k_i <= ~a), принимающих
участие в восстановлении секрета, через пробел: "
            k n)
    (tagbody try-again
       (setq ks (remove-duplicates (uiop:split-string (read-line) :separator "
")))
       (when (/= (length ks) k)
         (format t "~%Было введено некорректное количество сторон! Попробуйте
снова: ")
         (go try-again))
       (setq ks (mapcar #'(lambda (num?) (parse-integer num? :junk-allowed t))
ks))
       (when (some #'(lambda (num?) (or (null num?) (not (<= 1 num? n)))) ks)
         (format t "~%Было введено некорректное значение номера стороны!
Попробуйте ввести их снова: ")
         (go try-again))) (mapcar #'1- ks)))


(defun print-equations (cfs)
  (let ((k (aux:read-parse "k"))
        (len (length
              (write-to-string
               (apply #'max
                      (mapcar #'(lambda (cfsi)
                                  (apply #'max cfsi)) cfs)) :base 16))))
    (format t "~%Соответствующая система линейных уравнений для введённых
значений k_i:~2%")
    (do ((j 1 (1+ j))) ((< k j))
      (format t "~4t")
      (do ((s 1 (1+ s))) ((< k s))
        (format t (format nil "0x~v,'0x * x_~d + "
                          len (nth (1- s) (nth (1- j) cfs)) s)))
      (format t (format nil "0x~v,'0x (mod p) = 0;~%" len (nth k (nth (1- j)
cfs)))))))


(defun print-solution (solution)
  (let ((k (aux:read-parse "k")) (xs) (len))
    (format t "~%Решение приведённой системы уравнений:~%")
    (do ((j 0 (1+ j))) ((= k j) (setq xs (reverse xs)))
      (setq xs (cons (aref (bl-aux:array-slice solution j) k) xs)))
    (setq len (apply #'max (mapcar #'(lambda (num)
                                       (length (write-to-string num :base 16)))
xs)))
    (do ((j 0 (1+ j))) ((= k j))
      (format t (format nil "~%~4tx_~d = 0x~v,'0x;" (1+ j) len (nth j xs))))))


(defun recover-secret (ks)
```

```
  (let ((cfs (load-cfs "cfs")) (p (aux:read-parse "p"))
        (ds (mapcar #'parse-integer (uiop:read-file-lines "ds")))
        (solution) (secret))
    (setq cfs (mapcar #'(lambda (ki) (nth ki cfs)) ks)
          ds (mapcar #'(lambda (ki) (mod (- (nth ki ds)) p)) ks)
          cfs (mapcar #'(lambda (cfsi di) (append cfsi (list di))) cfs ds)
          solution (gauss:reduced-row-echelon (make-array (list (length cfs)
                                                                (length    (car
cfs)))
                                                          :initial-contents
cfs) p)
          secret (aref (bl-aux:array-slice solution 0) (length ks)))
    (print-equations cfs)
    (print-solution solution)
    (format t "~2%Значение восстановленного секрета: 0x~x = ~d.~%"
            (write-to-string secret :base 16) secret)))
```

```
(defpackage #:blakley
  (:use #:cl)
  (:export #:generate-point
           #:share-secret
           #:recover-secret))


(in-package #:blakley)


(defun stop () (read-line))


(defun generate-point ()
  (format t "~%Реализация (k, n)-пороговой схемы. Секрет M разделяется между n
сторонами так, чтобы
любые k из них могли восстановить секрет.~%")
  (bl-aux:get-n-k) (bl-aux:get-M)
    (bl-aux:gen-p) (bl-aux:gen-coords) t)


(defun share-secret ()
  (tagbody try-again
     (bl-aux:gen-cfs) (stop)
     (when (not (bl-aux:k-linear-independent?))
       (format t "~%~4tПолучившаяся система уравнений не является k-линейно
независимой. Коэффициенты будут
    перегенерированы.~2%")
       (go try-again))) t)


(defun recover-secret ()
  (format t "~%[2.1] -- Для восстановления секрета любым k сторонам необходимо
собраться вместе и из имеющихся долей
        секрета составить уравнения для отыскания точки пересечения
гиперплоскостей.")
  (let ((ks (bl-aux:get-ks)))
    (bl-aux:recover-secret ks)) t)
```