МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное образовательное
учреждение высшего образования
ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

Институт математики, механики и компьютерных наук
имени И. И. Воровича

Направление подготовки
02.03.02 — Фундаментальная информатика
и информационные технологии

# СХЕМЫ РАЗДЕЛЕНИЯ СЕКРЕТА С РЕАЛИЗАЦИЕЙ НА ЯЗЫКЕ HASKELL

Выпускная квалификационная работа
на степень бакалавра

Студента 4 курса
О. Н. Хритоненкова

Научный руководитель:
асс. каф. ИВЭ А. М. Пеленицын

Допущено к защите:
руководитель направления ФИИТ _____ В. С. Пилиди

Ростов-на-Дону
2016

# Содержание

# Введение

Защита секретной информации от потери и компрометации является одной из важнейших проблем современной криптографии. Метод создания нескольких копий данных и хранения их в разных местах повышает надежность при потере, но вместе с тем увеличивается вероятность компрометации ключа. Организовать надежное хранение конфиденциальной информации позволяют схемы разделения секрета (ключа), в которых секрет разделяется между участниками некоторой группы. При этом каждый участник получает долю секрета, а исходный секрет стирается. Для восстановления секрета требуется собрать определенное количество его долей.

Наиболее распространенными являются схемы, в которых количество долей, необходимых для восстановления секрета, может быть меньше общего количества участников. Такие схемы называются $(k, n)$–пороговыми, где $n$ — количество долей, на которые был разделен секрет, $k$ — количество долей, которые нужны для восстановления секрета. Таким образом, восстановить секрет способны любые $k$ и более сторон $(k \leq n)$. При этом любые $k - 1$ сторон восстановить секрет не смогут.

К важнейшим свойствам схем разделения секрета относятся идеальность, которая оценивает эффективность хранения данных в схеме, и совершенность, которая характеризует ее вычислительную стойкость. Вычислительная стойкость определяется объемом вычислений, требуемых для взлома данной схемы [1].

Существует несколько основных подходов к построению пороговых схем разделения секрета. Это основополагающие схемы Шамира и Блэкли, модулярный подход Миньотта и Асмута – Блума, а также основанная на решении систем уравнений схема Карнина – Грина – Хеллмана.

В данной работе были поставлены следующие задачи.

- Обоснование свойств совершенности и идеальности для схем разделения секрета Шамира и Блэкли.
- Анализ вычислительной сложности схем Шамира и Блэкли.
- Реализация схем Шамира и Блэкли на языке Haskell.

# 1. Предварительные сведения

В общем виде проблема разделения секрета была сформулирована еще в 1976 году Диффи и Хеллманом [2]. Идеи пороговых схем были независимо предложены Ади Шамиром [3] и Джорджем Блэкли [4] в 1979 году.

**Определение.** *Схема разделения секрета — криптографическая схема, позволяющая разделить секрет* $s$ *на* $n$ *частей. Части* $s_1, s_2, \ldots, s_n$*, называемые долями (частичными секретами), раздаются дилером* $D$ $n$ *участникам, при этом только определенные подмножества множества участников способны однозначно восстановить секрет. Такие подмножества называются разрешенными, остальные – запрещенными. Под структурой доступа* $\Gamma$ *понимается монотонное семейство разрешенных подмножеств* $A$*, т.е.*

$$\forall A, B : A \in \Gamma, A \subset B \Rightarrow B \in \Gamma$$

$\diamondsuit$

Частным случаем структуры доступа является $(k, n)$–пороговая структура доступа, в которой все подмножества из $k$ или более участников являются разрешенными. Такая структура доступа может быть реализована через $(k, n)$–пороговую схему разделения секрета.

**Определение.** $(k, n)$*–пороговая схема разделения секрета называется совершенной [5], если ни одно подмножество из менее чем* $k$ *участ-*

*ников не может получить никакой дополнительной информации о секрете, кроме априорной.* ♢

**Определение.** $(k, n)$*–пороговая схема разделения секрета называется идеальной, если размер доли секрета, предоставляемой участнику, равен размеру самого секрета.* ♢

# 2. Основные подходы к разделению секрета

## 2.1. Схема Шамира

### 2.1.1. Описание алгоритма

В основе предложенной А. Шамиром $(k, n)$–пороговой схемы разделения секрета лежит интерполяционная теорема Лагранжа.

**Теорема 1.** Для $k + 1$ пар чисел $(x_0, y_0)$, $(x_1, y_1)$, …, $(x_k, y_k)$, где все $x_i$ различны, существует единственный полином $L(x)$ степени не более $k$, для которого $L(x_i) = y_i$. Полином имеет вид

$$L(x) = \sum_{j=0}^{k} y_j l_j(x), \tag{1}$$

где базисные полиномы определяются по формуле:

$$l_j(x) = \prod_{i=0, j\neq i}^{k} \frac{(x - x_i)}{(x_j - x_i)} = \frac{(x - x_0)}{(x_j - x_0)} \cdots \frac{(x - x_{j-1})}{(x_j - x_{j-1})} \frac{(x - x_{j+1})}{(x_j - x_{j+1})} \cdots \frac{(x - x_k)}{(x_j - x_k)}.$$

△

Многочлен вида (1) получил название интерполяционного полинома Лагранжа.

Таким образом, для интерполяции полинома степени $k - 1$ требуется $k$ точек. Основная идея схемы Шамира состоит в том, что ин-

терполяция невозможна, если известно меньшее число точек [3]. Количество различных точек при этом ограничено размером числового поля, в котором ведутся расчеты.

**Алгоритм 1.** $(k, n)$–пороговая схема разделения секрета Шамира [3]

**Начало**: Дилер $D$ делит секрет $s$ на $n$ частей, каждый из $n$ участников получает долю секрета.

**Конец**: Если $k$ или более участников объединяются, они могут восстановить исходный секрет $s$.

1. *Разделение* Дилер $D$ знает некоторый секрет $s \geq 0$, который он делит среди $n$ участников следующим образом:
   a) $D$ выбирает случайное простое число $p > \max(s, n)$, которое задает конечное поле размера $p$, и кладет $a_0 = s$.
   b) $D$ случайно выбирает $k-1$ чисел $a_1, \ldots, a_{k-1}, 0 \leq a_j \leq p-1$, задающих многочлен над полем $\mathbb{Z}_p$: $f(x) = \sum_{j=0}^{k-1} a_j x^j$.
   c) $D$ вычисляет значения многочлена в $n$ различных точках (тени [6]): $\sigma_i = f(i) \bmod p, 1 \leq i \leq n$ и раздает каждому участнику долю секрета: тень $\sigma_i$ вместе с ее индексом $i$.
2. *Восстановление* Любая группа из $k$ или более участников объединяет свои доли. Эти доли представляют собой $k$ различных точек $(x_i, y_i) = (i, \sigma_i)$, позволяющих вычислить коэффициенты $a_j, 1 \leq j \leq k-1$ многочлена $f(x)$, используя интерполяционный полином Лагранжа. Секрет восстанавливается по формуле

$$L(0) = \sum_{i=1}^{k} y_i l_i(x) \bmod p = s, \quad \text{где} \quad l_i(x) = \prod_{j \neq i} \frac{x - x_j}{x_i - x_j} \bmod p.$$

### 2.1.2. Обоснование криптографических свойств

Схема Шамира является идеальной в предположении, что секрет выбран равновероятно в поле $\mathbb{Z}_p$. Поскольку $s < p$, справедлива фор-

мула Хартли:

$$m = \log_2 p,$$

где $m$ – количество бит.

Размер одной доли секрета складывается из суммы размера тени, передаваемой участнику, значение которой также находится в $\mathbb{Z}_p$, и размера ее индекса. Тогда размер секрета определяется как $B_s = m$ бит, размер одной доли секрета $B_{s_i} = B_{\sigma_i} + B_i = \log_2 p + \log_2 1 = m$ бит. Поскольку $B_s = B_{s_i}$, схема Шамира является идеальной.

Совершенность схемы следует из того, что секрет может быть восстановлен путем решения системы сравнений

$$\begin{cases} a_{k-1}j_1 + a_{k-2}j_1 + \ldots + a_1 j_1 + a_0 \equiv \sigma_{j_1} \pmod p \\ a_{k-1}j_2 + a_{k-2}j_2 + \ldots + a_1 j_2 + a_0 \equiv \sigma_{j_2} \pmod p \\ \ldots \\ a_{k-1}j_t + a_{k-2}j_t + \ldots + a_1 j_t + a_0 \equiv \sigma_{j_t} \pmod p \end{cases}$$

относительно коэффициентов $a_{k-1}, a_{k-2}, \ldots, a_0$, при этом $t < k$. Решением такой системы будет множество точек, которые лежат на гиперплоскости в $k$-мерном пространстве, следовательно, никакое значение секрета не будет отвергнуто как невозможное.

### 2.1.3. Анализ вычислительной сложности

Суммарная оценка сложности алгоритма складывается из оценок, полученных на этапе разделения и восстановления секрета.

На первом шаге алгоритма выбирается случайное простое число. Оценка этого шага зависит от выбора алгоритма, способного протестировать простоту случайно выбранного числа. Поскольку тестирование простоты требует существенных временных затрат, в качестве таких алгоритмов зачастую используются вероятностные тесты простоты числа. Так, в программной реализации, представленной в этой работе, использован тест Бейли–Померанца–Селфриджа–Уогстаффа.

На практике используются базы данных простых чисел с усредненным временем доступа, равным $O(1)$.

На втором шаге алгоритма необходимо выбрать $k-1$ коэффициентов для построения полинома над полем $\mathbb{Z}_p$. Сложность этого шага составит $O(k)$.

Количество итераций для вычисления теней на следующем шаге составляет $n$, причем каждое вычисление подразумевает внутренний цикл, проходящий по $k-1$ координатам. Таким образом, суммарная оценка этого шага составит $O(kn)$.

Для разделения долей $n$ участникам потребуется $n$ итераций. За время $O(n)$ участникам раздаются доли секрета.

Поскольку $k < n < kn$, оценка вычислительной сложности этапа разделения секрета составит $O(kn)$.

Для восстановления секрета используется построение интерполяционного полинома Лагранжа. Вычислительная сложность определяется подсчетом количества операций сложения, умножения и деления. Оценка этого шага составит $O(k^2)$ [7].

Поскольку $k^2 < kn$, общая оценка алгоритма составляет

$$O(kn),$$

где $k$ — минимально необходимое число участников для восстановления секрета, $n$ — общее число участников.

Затраты алгоритма по памяти анализируются в работе [8]. Число байт оперативной памяти, необходимой для хранения долей секрета, оценивается величиной $n|s| + O(|s|)$, где $|s|$ — верхний предел размерности для секрета $s$, характеристики $p$ и любого числа из $\mathbb{Z}_p$. При $k = n = 128$ подсчитано, что для восстановления секрета необходимо больше 112 кбайт оперативной памяти.

## 2.2. Схема Блэкли

### 2.2.1. Описание алгоритма

Основная идея схемы Блэкли, также называемой векторной схемой, в целом аналогична идее Шамира, за исключением того, что вместо многочлена используется уравнение плоскости в $k$–мерном пространстве. Для восстановления всех координат точки в $k$–мерном пространстве, принадлежащей нескольким гиперплоскостям, необходимо и достаточно знать уравнения $k$ таких плоскостей. То есть в двумерном пространстве нужны две пересекающиеся прямые, в трехмерном — три пересекающиеся в нужной точке плоскости и так далее.

**Определение.** *Гиперплоскостью называется подпространство с размерностью, на единицу меньшей, чем объемлющее пространство.* $\diamondsuit$

Разделяемым секретом в схеме Блэкли является одна из координат точки в $k$–мерном пространстве. Уравнения гиперплоскостей размерности $k-1$ служат долями секрета, раздаваемыми сторонам. Алгоритм разделения секрета по схеме Блэкли приведен на странице 10.

### 2.2.2. Обоснование криптографических свойств

Так же, как и в схеме Шамира, $s < p$ и, при условии того, что секрет выбран равновероятно в поле $\mathbb{Z}_p$, $B_s = \log_2 p = m$ бит. Каждая доля секрета состоит из $k$ чисел, выбранных в поле $\mathbb{Z}_p$. Размер одной доли секрета составит $B_{s_i} = k \log_2 p = km$ бит. Поскольку $B_s \neq B_{s_i}$, схема Блэкли не является идеальной.

Если собрано меньшее количество уравнений гиперплоскостей, их будет недостаточно для решения системы уравнений. Например, для $k-1$ участников результатом решения системы будет прямая в $k$–мерном пространстве. Это означает, что секрет может равновероятно принимать любое значение из поля $\mathbb{Z}_p$. В этом случае никакое

**Алгоритм 2.** $(k, n)$–пороговая схема разделения секрета Блэкли [4]

**Начало**: Дилер $D$ делит секрет $s$ на $n$ частей, каждый из $n$ участников получает долю секрета.

**Конец**: Если $k$ или более участников объединяются, они могут восстановить исходный секрет $s$.

1. *Разделение* Дилер $D$ знает некоторый секрет $s \geq 0$, который он делит среди $n$ участников следующим образом:

   a) $D$ выбирает случайное большое простое число $p > \max(s, n)$, которое задает конечное поле размера $p$, и кладет $b_1 = s$.

   b) $D$ случайно выбирает $k-1$ чисел $b_2, \ldots, b_k, 0 \leq b_j \leq p-1$, вместе с $b_1$ задающих точку в $k$–мерном пространстве, первая координата которой является секретом.

   c) Для каждого из $n$ участников $D$ случайно выбирает $k$ чисел $a_{1_i}, \ldots, a_{1_k}, 0 \leq a_{1_i} \leq p-1$. Так как уравнение плоскости имеет вид $a_{1_i}x_1 + a_{2_i}x_2 + \ldots + a_{k_i}x_k + d_i = 0$, для каждого участника необходимо вычислить коэффициент $d_i$:

   $$d_i = -(a_{1_i}s + a_{2_i}b_2 + \ldots + a_{k_i}b_k) \mod p$$

   При этом любые $k$ уравнений должны быть линейно независимы. $D$ раздает участникам доли — набор коэффициентов, задающих уравнение гиперплоскости.

2. *Восстановление* Любая группа из $k$ или более участников объединяет свои доли. Для отыскания точки пересечения гиперплоскостей и восстановления секрета необходимо решить систему линейных уравнений

   $$\begin{cases} (a_{1_1}x_1 + a_{2_1}x_2 + \ldots + a_{k_1}x_k + d_1) \mod p = 0 \\ (a_{1_2}x_1 + a_{2_2}x_2 + \ldots + a_{k_2}x_k + d_2) \mod p = 0 \\ \ldots \\ (a_{1_k}x_1 + a_{2_k}x_2 + \ldots + a_{k_k}x_k + d_k) \mod p = 0 \end{cases}$$

   Решение системы дает точку в $k$–мерном пространстве, первая координата которой является исходным секретом.

значение секрета не будет отвергнуто как невозможное. Значит, схема Блэкли является совершенной.

### 2.2.3. Анализ вычислительной сложности

Так же, как и в схеме разделения секрета Шамира, оценка первого шага алгоритма зависит от выбранного метода проверки простоты числа. В общем случае оценка этого шага составит $O(1)$.

Сложность второго шага данного алгоритма в точности совпадает со сложностью второго шага схемы Шамира. При выборе $k-1$ чисел оценка составит $O(k)$.

На следующем шаге для каждого из $n$ участников вычисляется коэффициент $d_i$, причем на каждой итерации используется набор из $k$ случайно выбранных чисел. Оценка на этом шаге составит $O(kn)$.

Для разделения долей $n$ участникам потребуется $n$ итераций. За время $O(n)$ участникам раздаются доли секрета.

Поскольку $k < n < kn$, оценка вычислительной сложности этапа разделения секрета составит $O(kn)$.

Для восстановления секрета необходимо решить систему линейных алгебраических уравнений. Поскольку секретом является первая координата точки, полученной в результате решения, лучшим вариантом будет метод Крамера [8]. Для восстановления секрета потребуется вычислить 2 определителя матриц размерности $k \times k$. При использовании метода Гаусса для вычисления определителей оценка составит $2O(k^3)$, где $O(k^3)$ – вычислительная сложность метода Гаусса [9]. Таким образом, оценка этапа восстановления секрета составит $O(k^3)$.

Общая оценка алгоритма составляет

$$O(kn) + O(k^3),$$

где $k$ — минимально необходимое число участников для восстановления секрета, $n$ — общее число участников.

В работе [8] проведен анализ ресурсоемкости вычислений. Число байт оперативной памяти, необходимой для разделения секрета на доли, оценивается величиной $nk|s|$, где $|s|$ — верхний предел размерности для секрета $s$, характеристики $p$ и любого числа из $\mathbb{Z}_p$. Аналогичная оценка получена и для восстановления секрета. При $k = n = 128$ подсчитано, что для операций восстановления и разделения секрета необходимо свыше 2 Мбайт оперативной памяти.

# 3. Реализация схем Шамира и Блэкли на языке Haskell

Для хранения секрета используется целочисленный тип **`Integer`**, размер которого ограничен оперативной памятью компьютера.

Выбранное случайное число $p$ проверяется на простоту с использованием теста Бейли–Померанца–Селфриджа–Уогстаффа. Вероятностные алгоритмы реализованы в ряде Haskell-библиотек, например, в библиотеке The arithmoi [10]. В модуле `Math.NumberTheory.Primes.Testing` данной библиотеки определена функция

```
isPrime :: Integer -> Bool
```

Она выполняет проверку числа на простоту с использованием теста Бейли–Померанца–Селфриджа–Уогстаффа.

## 3.1. Реализация схемы Шамира

Полиномы удобно задавать списком их коэффициентов, расположенных в порядке возрастания степеней. Так, многочлен $1+3x+5x^3$ может быть представлен в виде списка `[1,3,0,5]`.

```
type Polynomial = [Integer]
```

Таким образом, генерация случайного полинома степени $k-1$ сводится к генерации списка его коэффициентов, каждый из которых лежит в поле $\mathbb{Z}_p$ (листинг 3.1).

**Листинг 3.1.** Случайный многочлен над полем $\mathbb{Z}_p$

```
randomList :: Integer -> Integer -> IO [Integer]
randomList k p = do
        listZp <- generate 0 (p-1)
        return $ takeN (k-1) listZp
    where
        generate :: Random a => a -> a -> IO [a]
        generate a b = fmap (randomRs (a,b)) newStdGen
```

Здесь функция `takeN` (листинг 3.2) представляет собой вариант функции **`take`** из стандартной библиотеки языка для чисел типа **`Integer`**. Она возвращает указанное количество элементов из начала списка.

**Листинг 3.2.** Функция take для типа Integer

```
takeN :: Integer -> [a] -> [a]
takeN n list = take (fromIntegral n) list
```

Функция `evalPoly` (листинг 3.3) вычисляет значение полинома в точке. Вычисление производится по модулю $p$. Список `temp` содержит пары, состоящие из коэффициента монома и его степени, образованных при помощи стандартной функции **`zip`**.

**Листинг 3.3.** Значение полинома в точке

```
evalPoly :: Polynomial -> Integer -> Integer -> Integer
evalPoly poly x p = mod (sum [snd a * x ^ (fst a) | a <- temp]) p
  where
        temp = zip [0..] poly
```

Функция `evalShares` (листинг 3.4) вычисляет значения полинома для точек из переданного списка `points`, соединяя каждое полученное значение с его индексом.

**Листинг 3.4.** Вычисление долей секрета

```
evalShares
    :: Polynomial -> [Integer] -> Integer -> [(Integer,Integer)]
evalShares poly points p
    = zip [1..] [evalPoly poly x p | x <- points]
```

**Листинг 3.5.** Множество всех подмножеств заданной длины

```
getSubsets :: Integer -> [a] -> [[a]]
getSubsets k = filterSubsets k . getSubsets'
 where
  getSubsets' [] = [[]]
  getSubsets' (x:xs) = s ++ map (x:) s where s = getSubsets' xs

  filterSubsets :: Integer -> [[a]]-> [[a]]
  filterSubsets k [] = []
  filterSubsets k (x:xs)= if (fromIntegral (length x) == k) then
     (x:filterSubsets k xs) else filterSubsets k xs
```

На этом завершается этап разделения секрета. Любая группа из $k$ или более участников может его восстановить, тогда как любая группа из менее чем $k$ участников сделать этого не сможет.

Функция `getSubsets` (листинг 3.5) принимает на вход целое число $k$ и список элементов типа `a` и возвращает список, элементами которого являются все подсписки списка типа `a` длины $k$.

Полученный список может быть использован в качестве структуры доступа.

Наконец, функция `evalLagrange` (листинг 3.6) позволяет восстановить секрет, выполняя построение интерполяционного полинома Лагранжа и используя при этом ряд вспомогательных функций.

Функция `l` строит базисные полиномы.

Для целочисленного деления в конечном поле используется функция `f`, которая выполняет умножение на обратный по модулю делителя элемент.

Для нахождения обратного по модулю делителя элемента реализована функция `inverse`, которая использует расширенный алгоритм Евклида (`eGCD`), позволяющий найти обратный по модулю элемент.

**Листинг 3.6.** Восстановление секрета

```
evalLagrange :: Integer -> [(Integer, Integer)] -> Integer
evalLagrange p shares = mod (sum $ map g shares) p
  where
    g (c, y) = y * l c
    l i = product [f x | (x, _) <- shares, x /= i]
      where
        f x     = (-x) * inverse (i - x) p
    inverse a = fst . eGCD a
    eGCD a b
      | b == 0  = (1, 0)
      | otherwise = (t, s - q * t)
        where
          (q, r) = divMod a b
          (s, t) = eGCD b r
```

### 3.1.1. Пример работы программы

Для демонстрации работы программы используется функция `shamir`, которая принимает на вход разделяемый секрет $s$, параметры $k$ и $n$, определяющие $(k, n)$–пороговую схему, а также количество $k'$ участников, пытающихся восстановить секрет, и выводит на экран результат разделения и восстановления секрета при заданных параметрах.

Для $(3, 4)$–пороговой схемы Шамира три участника восстанавливают секрет (листинг 3.7).

**Листинг 3.7.** (3,4)–пороговая схема Шамира с тремя участниками

```
ghci> shamir 9895 3 4 3
secret = 9895
shares = [(1,243),(2,1288),(3,2297),(4,3270)]
secret = 9895
```

Два участника не могут восстановить секрет (листинг 3.8).

**Листинг 3.8.** (3,4)–пороговая схема Шамира с двумя участниками

```
ghci> shamir 157693 3 4 2
secret = 157693
shares = [(1,282708),(2,128374),(3,165342),(4,393612)]
secret = 437042
```

## 3.2. Реализация схемы Блэкли

Для получения списков случайных чисел $b_2, \ldots, b_k$ и $a_{1_i}, a_{2_i}, \ldots, a_{k_i}$ удобно использовать функцию `randomList`, описанную в подразделе 3.1.

Функция `getDiList` принимает на вход количество разрешенных участников $k$, секрет $s$, простое число $p$, а также список коэффициентов $b_2, \ldots, b_k$ и $a_{1_i}, a_{2_i}, \ldots, a_{k_i}$ возвращает список коэффициентов $d_1, \ldots, d_n$ (листинг 3.9). При этом используются две вспомогательные функции: `getDi` (листинг 3.10) и `sublist` (листинг 3.12).

**Листинг 3.9.** Вычисление списка коэффициентов $d_i$

```
getDiList :: Integer -> Integer -> Integer ->
  [Integer] -> [Integer] -> [Integer]
getDiList k s p blist alist
  = map (getDi s p blist) (sublist k alist)
```

Функция `sublist` разбивает список из $kn$ случайных коэффициентов $a_{j_i}$ на $n$ подсписков длины $k$. Далее для каждого такого подсписка функция `getDi` вычисляет коэффициент $d_i$, используя стандартную функцию **`zipWith`** и левую свертку **`foldl`**. Поскольку $d_i$ могут быть отрицательными, для получения остатка от деления используется функция **`rem`**.

**Листинг 3.10.** Вычисление коэффициента $d_i$

```
getDi :: Integer -> Integer -> [Integer] -> [Integer] -> Integer
getDi m p blist alist = rem ((head alist * m) +
  foldl (+) 0 (zipWith (*) (tail alist) blist)) p
```

Помимо функции `takeN`, описанной в подразделе 3.1, определена функция `dropN` (листинг 3.11), представляющая собой вариант функции **`drop`** из стандартной библиотеки языка для чисел типа **`Integer`**. Она удаляет указанное количество элементов из начала списка.

**Листинг 3.11.** Функция **`drop`** для типа **`Integer`**

```
dropN :: Integer -> [a] -> [a]
dropN n l = drop (fromIntegral n) l
```

**Листинг 3.12.** Все подсписки длины $n$

```
sublist :: Integer -> [a] -> [[a]]
sublist n ls
    | n <= 0 || null ls = []
    | otherwise = takeN n ls:sublist n (dropN n ls)
```

Функция `getEquations` (листинг 3.13) принимает на вход $k$, список из $n$ коэффициентов $a_{j_i}$ и $n$ коэффициентов $d_i$ и возвращает список из $n$ списков, каждый из которых содержит коэффициенты $a_{1_i}, a_{2_i}, \ldots, a_{k_i}$, $d_i$ — доли секрета, которые раздаются участникам.

**Листинг 3.13.** Разделение секрета на доли

```
getEquations :: Integer -> [Integer] -> [Integer] -> [[Integer]]
getEquations k alist dlist = map (change) temp
  where
    temp = zipWith (:) dlist (sublist k alist)
    change (x:xs) = xs ++ [x]
```

На этом завершается этап разделения секрета.

Для восстановления секрета потребуется вычислить два определителя матриц размерности $k \times k$, где $k$ — количество участников, пытающихся восстановить секрет. При этом все вычисления производятся в конечном поле.

Входными параметрами функции `determinant` (листинг 3.14) являются список списков, описывающий матрицу, и простое число $p$, по модулю которого выполняются вычисления.

Здесь функция `returnWithout` возвращает матрицу без верхней строки и $i$ столбца.

**Листинг 3.14.** Подсчет определителя матрицы

```
determinant :: [[Integer]] -> Integer -> Integer
determinant [[x]] p = x
determinant mat p = sumrem [multrem ((-1)^i*x)
  (determinant (returnWithout i mat) p) p |
  (i, x) <- zip [0..] (head mat)] p
  where
    multrem a b p = (a * b) `rem` p

    sumrem [] p = 0
    sumrem [x] p = rem x p
    sumrem (x:y:xs) p = (rem (x+y) p) + sumrem xs p

    returnWithout i mat = removeCols i (tail mat)

    removeCols _ [] = []
    removeCols i (r:rs) = (left ++ (tail right))
     : removeCols i rs
         where (left, right) = splitAt i r
```

### 3.2.1. Пример работы программы

Для демонстрации работы программы используется функция `blakley`, которая принимает на вход разделяемый секрет $s$, параметры $k$ и $n$, определяющие $(k, n)$–пороговую схему, а также количество $k'$ участников, пытающихся восстановить секрет, и выводит на экран результат разделения и восстановления секрета при заданных параметрах.

Для $(3, 4)$–пороговой схемы Блэкли три участника восстанавливают секрет (листинг 3.15).

Два участника не могут восстановить секрет (листинг 3.16).

**Листинг 3.15.** (3,4)–пороговая схема Блэкли с тремя участниками

```
ghci> blakley 105 3 4 3
secret = 105
shares = [[280,218,300,30],[127,270,92,139],
[176,16,222,66],[271,254,140,37]]
secret = 105
```

**Листинг 3.16.** (3,4)–пороговая схема Блэкли с двумя участниками

```
ghci> blakley 78 3 4 2
secret = 78
shares = [[70,103,84,11],[52,163,123,19],
[107,69,147,172],[154,20,12,44]]
secret = 34
```

## Заключение

В ходе данной работы было получено обоснование свойств совершенности и идеальности для схем разделения секрета Шамира и Блэкли, проведен анализ их вычислительной сложности, а также представлена реализация на языке программирования Haskell.

Полный исходный код доступен по адресу [11].

## Список литературы


1. *Пилиди В. С.* Криптография. Вводные главы. — ЮФУ, 2009.
2. *Diffie W.*, *Hellman M. E.* New Directions in Cryptography // IEEE Transactions on Information Theory. — 1976. — URL: http://www-ee.stanford.edu/~hellman/publications/24.pdf.
3. *Shamir A.* How to share a secret // Communications of the ACM. — 1979. — Т. 22, № 11. — С. 612—613. — ISSN 0001-0782. — DOI: 10.1145/359168.359176.

4. *Blakley G. R.* Safeguarding cryptographic keys // Proceedings of the 1979 AFIPS National Computer Conference. — 1979. — С. 313—317. — DOI: 10.1109/AFIPS.1979.98.
5. *Pomerance C.* Advances in Cryptology — CRYPTO '87. Т. 293. — Springer, 1988.
6. *Шнайер Б.* Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си. — Триумф, 2002. — С. 588.
7. *Ващенко Г. В.* Вычислительная математика. Основы алгебраической и тригонометрической интерполяции. — СибГТУ, 2008. — С. 40.
8. Сравнительный анализ стойкости некоторых классов схем разделения секрета. — URL: http://master.cmc.msu.ru/files/master2013_1_pianov.pdf (дата обр. 18.04.2016).
9. *Вержбицкий В. М.* Вычислительная линейная алгебра. — Директ-Медиа, 2013. — С. 299.
10. The arithmoi package. — URL: http://hackage.haskell.org/package/arithmoi-0.4.2.0 (дата обр. 31.05.2016).
11. Shamir-and-Blakley-Secret-Sharing. — URL: http://github.com/coastline7/Shamir-and-Blakley-Secret-Sharing (дата обр. 20.06.2016).