Ссылка на статью:
// Ученые записки УлГУ. Сер. Математика и информационные технологии. 2023, № 1, с. 97–147.
Поступила: 08.01.2023
Окончательный вариант: 26.01.2023

УДК 519.7

О реализации некоторых совершенных схем разделения секрета

Рацеев С. М.

ratseevsm@mail.ru
УлГУ, Ульяновск, Россия

В работе приводится программная реализация схемы разделения секрета Шамира над конечным полем характеристики два, схемы на основе равновесных двоичных кодов и схемы Ито—Саито—Нишизеки. Первые две схемы являются пороговыми схемами разделения секрета, третья — схемой с произвольной структурой доступа. Важность этих схем заключается в том, что они являются совершенными. Работа носит учебно-методический характер и может помочь с программной реализацией схем разделения секрета.

Ключевые слова: *схема разделения секрета, структура доступа, схема Шамира, схема Ито—Саито—Нишизеки*

Содержание

Введение

Одной из самых популярных пороговых схем разделения секрета является схема Шамира [1]. Данная схема, в частности, обладает свойством совершенности (доли любой неправомочной коалиции не позволяют получить какой-либо информации о секрете), идеальности (число битов, содержащихся в каждой доле секрета, равно числу битов, содержащихся в самом секрете), расширяемости (число владельцев долей секрета n может быть в любой момент увеличено, при этом количество частей секрета t, необходимых для восстановления секрета, останется неизменным) и т. д. Схема на основе равновесных двоичных кодов идеальной не является, хотя и обладает свойством совершенности. Данная схема позволяет в некоторых ситуациях восстановить секрет без использования дополнительных вычислительных устройств, например, персонального компьютера (ПК), поэтому заслуживает внимания. Также интересной схемой разделения секрета для произвольной структуры доступа является схема Ито—Саито—Нишизеки, которая тоже обладает свойством совершенности. Свойство совершенности играет важную роль в криптосистемах. Это свойство, другими словами, означает, что эта криптосистема не зависит от вычислительных возможностей криптоаналитика. С учетом активного поиска постквантовых криптосистем [2] криптосистемы со свойством совершенности заслуживают особого внимания. Заметим, что совершенные схемы разделения секрета можно строить и на основе корректирующих кодов [3–6].

Работа носит учебно-методический характер и является продолжением работ [7, 8]. В ней приводятся программные реализации схемы разделения секрета Шамира, схемы на основе равновесных двоичных кодов и схемы Ито—Саито—Нишизеки. В роли секрета может выступать произвольный файл. Для достижения свойства идеальности для схемы Шамира будут использованы конечные поля характеристики два. Для увеличения скорости работы программы будут задействованы таблицы степеней образующего элемента и таблицы дискретных логарифмов по основанию образующего элемента поля. Приведенные программные реализации с пояснениями могут помочь в учебных целях при написании лабораторных работ, курсовых и дипломных работ, связанных с данной тематикой.

1. Вычислительная сложность схемы разделения секрета Шамира

Пусть $s \in GF(q)$ — секрет, который нужно разделить на n участников с порогом t. Не будем учитывать вычислительную сложность генерации коэффициентов a_i многочлена Лагранжа $L(x)$ степени не более $t-1$, так как это зависит от выбранного генератора. При вычислении доли $y_i = L(x_i)$ с помощью схемы Горнера понадобится $t-1$ операций умножения и столько же операций сложения. Так как участников n, то понадобится $n(t-1)$ операций сложения и столько же операций сложения. Поэтому вычислительная сложность вычисления n долей секрета составляет $O(nt)$ операций. Пусть l — совокупность секретов $s_1, \ldots, s_l$, которые нуж-

но разделить на n участников (например, это файл из совокупности l секретов). Тогда общая вычислительная сложность разделения секрета составит $O(ntl)$ операций.

В нашей программе будут использованы таблицы степеней образующего элемента и таблицы дискретных логарифмов. Поэтому операция умножения элементов поля заменится на операцию сложения дискретных логарифмов. При этом вычислительная сложность составит $O(ntl)$ операций.

При восстановлении секрета s на основе t долей $(x_1, y_1),\ldots,(x_t, y_t)$ (здесь x_i, y_i обозначены как и ранее, но это не значит, что (x_i, y_i) — доля i-го исходного участника, теперь это обозначение доли i-го собравшегося участника) используется формула

$$S = L(0) = \sum_{i=1}^{t} y_i \prod_{\substack{1\le j\le t \\ j\ne i}} \frac{x_j}{x_j - x_i}.$$

Для вычисления $\frac{x_j}{x_j - x_i}$ на основе таблиц степеней и дискретных логарифмов вычислительная сложность составляет $O(t)$ операций. Учитывая сумму, в которой t слагаемых, вычислительная сложность составит $O(t^2)$ операций. Пусть l — исходная совокупность секретов $s_1,\ldots,s_l$, которые были разделены на n участников. Тогда общая вычислительная сложность восстановления секрета составит $O(t^2 l)$ операций.

2. Программная реализация схемы разделения секрета Шамира

В следующей программе происходит разделение произвольного файла-секрета на n долей с порогом t над полем $GF(2^8)$, которое строится с помощью примитивного многочлена $p(x) = x^8 + x^7 + x^5 + x^3 + 1$. Так как число ненулевых элементов поля $GF(2^8)$ равно 255, то число участников n не должно превышать это число. Если $n > 255$, то нужно другое поле ($GF(2^{16})$, $GF(2^{24})$ и т. п.). Файлы-доли секрета запишем в директорию dirname, которая задается в программе. Если такой директории на диске нет, то она будет создана. Имя файла-секрета задается в строковой переменной fsecret_name (файл может быть произвольным). При вызове функции Secret_sharing_file файл-секрет будет разделен на файлы-доли с именами, которые заданы в константе FRACTION_NAME (вдобавок к ним будут приписаны номера участников). В начало каждого файла-доли будет записан соответствующий номер участника, поэтому размер каждого файла-доли будет отличаться на размер файла-секрета на 1 байт (номера участников i, $1 \le i \le n \le 255$, помещаются в один байт). Заметим, что номера участников можно не записывать в файлы-доли (тогда размеры файла-секрета и файлов-долей совпадут), а номера собравшихся участников извлекать из имен файлов-долей. Но мы выберем первый вариант.

Для восстановления файла-секрета на основе не менее t файлов-долей вызывается функция Restoring_secret_file. Перед ее вызовом можно закомментировать вызов функции Secret_sharing_file, а также можно оставить в директории dirname не менее t файлов-долей

для успешного восстановления файла-секрета. При вызове функции Restoring_secret_file будет осуществлен просмотр всех имен файлов в директории dirname и прочитано содержимое тех из них, имена которых совпадают с названием заданного в константе FRACTION_NAME имени файла-доли (без учета добавленных номеров участников к именам файлов). Если число файлов-долей не менее t, то файл-секрет будет восстановлен с полным именем fsecret_name2.

В программе для работы с файлами используются буферы обмена. Заметим, что они позволяют ускорить работу с файлами порой в десятки раз.

В программе присутствуют, в частности, следующие функции.

Функция

```
uint8 Value_of_polynomial(const uint8 *a, int n, uint8 x0)
```

вычисляет значение многочлена $a(x)$ степени не выше n в точке $x = x_0$ с помощью схемы Горнера.

Функция

```
int Secret_sharing(uint8 s, uint8 *y, int n, int t)
```

вычисляет n долей секрета s с порогом t, которые записываются в массив долей y.

Функция

```
uint8 Restoring_secret(uint8 *x, uint8 *y, int t)
```

по номерам и долям t собравшихся участников, которые находятся в массивах x и y соответственно, вычисляет значение секрета s.

Функция

```
int Secret_sharing_file(char *fname, char *dirname, int n, int t)
```

разделяет файл-секрет с именем fname на n участников с порогом t и файлы-доли секрета записывает в директорию с именем dirname.

Функция

```
int Restoring_secret_file(char *fname, char *dirname, int t)
```

пробегает по директории с именем dirname, отыскивает в ней файлы-доли не менее t участников и в случае успешного нахождения не менее t файлов-долей восстанавливает файл-секрет, который записывает в файл с именем fname.

Для работы с файлами-долями файла-секрета нужно уметь работать с директориями (иными словами, папками, каталогами и т. п.). Нам понадобится структурный тип данных struct dirent. Это тип структуры, используемый, чтобы возвратить информацию относительно входов в директорию. Одним из полей данной структуры является поле char *d_name, значением которого является имя директории/файла. Тип данных DIR представляет поток директории. Для работы с директорией необходимо ее открыть с помощью функции opendir, которая имеет следующий прототип: DIR * opendir (const char *dirname). Функция struct

dirent *readdir(DIR *dir) возвращает указатель на структуру struct dirent, представляющую следующую запись директории в потоке директорий (в частности, файлов), указанного в dir. Функция возвращает NULL по достижении последней записи в потоке директорий или если произошла ошибка. Для примера ниже представлена простая программа для просмотра содержимого диска d.

```
#include <stdio.h>
#include <dirent.h>
#include <sys/types.h>

int main()
{
    struct dirent *entry;
    DIR *dir = opendir("d:\\");
    if (dir == NULL)
        return 1;
    while ((entry = readdir(dir)) != NULL)
        puts(entry->d_name);
    closedir(dir);
    return 0;
}
```

Библиотека dirent.h в первую очередь предназначена для linux. Но работает и с windows. Она поставляется вместе с такими компиляторами, как GCC (Cross-platform), MinGW (Microsoft Windows) и т. д., поэтому программа работает сразу. Если, например, нужно запустить представленную программу в Visual studio, то можно скачать из интернета (например, https://github.com/tronkko/dirent) библиотеку dirent.h и поместить в папку include в Visual studio. Все представленные в этой работе программы с легкостью запускались и работали в C-FREE. Заметим, что для отличия файлов и директорий в программах можно использовать struct stat statbuf.

Ниже приводится программа для разделения и восстановления файла-секрета.

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<string.h>
#include <dirent.h>
#include <dir.h>
#include <sys/types.h>

// Параметры конечного поля
#define N 8 // поле GF(2^N)
```

```
#define NMAX 255 // число 2^N - 1 - максимальное N-битное число
#define P 0x1a9 // многочлен p(x)=x^8+x^7+x^5+x^3+1 в 16-ичном виде
#define ALPHA 2 // 00000010 - корень многочлена p(x)

#define FRACTION_NAME "fraction" // имена файлов долей секрета
#define FILE_LEN 512 // максимальная длина полного имени файла

#define BUFSIZE 4048 // размер буфера обмена

typedef unsigned char uint8;
typedef unsigned short uint16;
typedef unsigned long long uint64;

// степени образующего элемента
uint8 deg_alpha[2*NMAX + 1];
// таблица дискретных логарифмов по основанию образующего элемента
uint8 log_alpha[NMAX + 1];

// Произведение элементов a и alpha
uint8 Product_alpha(uint8 a)
{
    uint16 c = a;
    c <<= 1;
    if ((c >> N) & 1)
         c ^= P;
    return c;
}

// Произведение элементов поля a и b
uint8 Product(uint8 a, uint8 b)
{
    uint16 buf; // для промежуточных вычислений
    uint8 rez; // результат произведения
    buf = b;
    rez = 0;
    while(a)
    {
        if (a & 1)
            rez ^= buf;
        buf <<= 1;
        if ((buf >> N) & 1)
```

```
            buf ^= P;
        a >>= 1;
    }
    return rez;
}

// Размер файла в байтах
uint64 Size(const char *fileName)
{
   FILE *f;
   uint64 size;
   if ((f = fopen(fileName, "rb")) == NULL)
      return 0;
   fseek(f, 0, SEEK_END); // перемещаем указатель в конец файла
   size = ftell(f);       // считываем текущую позицию указателя
   fclose(f);
   return size;
}

// Вычисление значения многочлена a(x) степени не выше n в точке x=x0
// с помощью схемы Горнера
uint8 Value_of_polynomial(const uint8 *a, long n, uint8 x0)
{
    uint8 y;
    long i;
    if (x0 == 0)
        return a[0];
    y = a[n];
    for(i = n - 1; i >= 0; i--)
        y = (y ? (a[i] ^ deg_alpha[ log_alpha[y] + log_alpha[x0] ]) : a[i]);
    return y;
}

// Разделение секрета s на n долей с порогом t,
// которые записываются в массив y
int Secret_sharing(uint8 s, uint8 *y, long n, long t)
{
    long i, j;
    uint8 *a;  // коэффициенты многочлена Лагранжа

    a = (uint8 *)malloc(t * sizeof(*a));
```

```
    if (a == NULL)
        return 0;
    a[0] = s;
    // Здесь для примера используется небезопасный генератор rand()
    for(i = 1; i < t; i++)
        a[i] = rand() % NMAX;

    // Вычисление долей секрета секрета
    for(i = 0; i < n; i++)
        y[i] = Value_of_polynomial(a, t-1, i+1);

    free(a);
    return 1;
}

// Восстановление секрета s по t долям (x[0],y[0]),...,(x[t-1],y[t-1])
uint8 Restoring_secret(uint8 *x, uint8 *y, long t)
{
    long i, j;
    uint8 s, b, prod;

    // Восстановление секрета с использованием дискретных логарифмов
    s = 0;
    for(i = 0; i < t; i++)
    {
        prod = 1;
        if (y[i] != 0)
        {
            for(j = 0; j < t; j++)
            {
                if (j != i)
                {
                    // b = x_j/(x_j-x_i)
                    b = deg_alpha[ log_alpha[x[j]] + NMAX - log_alpha[x[j] ^ x[i]] ];
                    // prod - произведение таких b при всех j, не равных i
                    prod = deg_alpha[ log_alpha[prod] + log_alpha[b] ];
                }
            }
            s ^= deg_alpha[ log_alpha[y[i]] + log_alpha[prod] ];
        }
    }
```

```
    return s;
}

// Выделение памяти для двумерного динамического массива-буфера
// размера n на BUFSIZE для файлов-долей секрета
uint8 **Allocate(long n)
{
    uint8 **a;
    long i;
    a = (uint8 **)malloc(n*BUFSIZE*sizeof(uint8) + n*sizeof(uint8 *));
    if (a != NULL)
        for(i = 0; i < n; i++)
             a[i] = (uint8 *)(a + n) + i*BUFSIZE;
    return a;
}

// Разделение файла-секрета fname на n участников с порогом t.
// Файлы-доли секрета будут помещены в директорию dirname
int Secret_sharing_file(char *fname, char *dirname, long n, long t)
{
    FILE *fsecret, // файл-секрет
        **f; // файлы-доли секрета
    char name[FILE_LEN], st[FILE_LEN];
    long i, j, len, count;
    uint8 s; // секрет
    uint8 *y; // доли секрета
    uint8 *bufs, **bufy; // буферы обмена для файла-секрета и файлов-долей

    // Выделяем место для буферов обмена
    bufs = (uint8 *)malloc(BUFSIZE*sizeof(*bufs));
    bufy = Allocate(n);
    if (bufs == NULL || bufy == NULL)
        return 3;

    // Создаем директорию для файлов-долей
    mkdir(dirname);
    // Открываем файл-секрет для чтения
    if ((fsecret = fopen(fname, "rb")) == NULL)
        return 10;
    // В переменную name будут по очереди записаны полные имена файлов-долей
    // секрета. Изначально в строковую переменную  name записываем полное
```

```
// название директории dirname и добавляем в конец "\fraction".
strcpy(name, dirname);
strcat(name, "\\");
strcat(name, FRACTION_NAME);
len = strlen(name);
// Динамический массив файлов-указателей-долей секрета
f = (FILE **)malloc(n * sizeof(*f));
// Динамический массив для долей секрета
y = (uint8 *)malloc(n * sizeof(*y));
if (f == NULL || y == NULL)
    return 20;
// Открываем файлы-доли секрета для записи
for(i = 0; i < n; i++)
{
    itoa(i + 1, st, 10); // переводим номер участника в строку st
    strcpy(name + len, st); // добавляем номер к названию файла-доли секрета
    f[i] = fopen(name, "wb"); // открываем файл-долю для записи
    if (f[i] == NULL)
        return 30;
}

// Запишем номера участников в начало каждого файла с долями
for(i = 1; i <= n; i++)
    fwrite(&i, sizeof(uint8), 1, f[i-1]);

// Разделяем файл-секрет на доли, которые записываем в файлы-доли секрета,
// используя для ускорения работы буфер обмена bufs
while (count = fread(bufs, sizeof(*bufs), BUFSIZE, fsecret))
{
    for(j = 0; j < count; j++)
    {
        // Вычисляем доли y секрета bufs[j]
        Secret_sharing(bufs[j], y, n, t);
        // Записываем доли секрета bufs[j] в буфер bufy
        // для файлов-долей секрета
        for(i = 0; i < n; i++)
            bufy[i][j] = y[i];
    }
    // Накопленные в буфере bufy доли секрета записываем в файлы-доли
    for(i = 0; i < n; i++)
       fwrite(bufy[i], sizeof(uint8), count, f[i]);
```

```
    }

    fclose(fsecret);
    for(i = 0; i < n; i++)
        fclose(f[i]);
    free(y); free(f); free(bufs); free(bufy);
    return 0;
}

// Восстановление файла-секрета по t файлам-долей
int Restoring_secret_file(char *fname, char *dirname, int t)
{
    FILE *fsecret, // файл-секрет
        **f; // файлы-доли секрета
    struct dirent *entry; // для чтения файлов-долей секрета из директории
    DIR *dir; // указатель-директория
    char name[FILE_LEN], st[FILE_LEN];
    long i, len;
    uint8 *x, *y;
    uint64 j, k, size, q, r;
    uint8 *bufs, **bufy; // буферы обмена для файла-секрета и файлов-долей

    // Выделяем место для буферов обмена
    bufs = (uint8 *)malloc(BUFSIZE*sizeof(*bufs));
    bufy = Allocate(t);
    if (bufs == NULL || bufy == NULL)
        return 3;

    // Открываем директорию, откуда будем брать файлы-доли секрета
    if ((dir = opendir(dirname)) == NULL)
        return 5;
    // В переменной name будут по-очереди записаны полные имена файлов-долей секрета
    strcpy(name, dirname);
    strcat(name, "\\");
    len = strlen(name);
    // Создаем динамические массивы:
    // x - номера участников,
    // y - доли секрета,
    // f - файловые указатели-доли секрета
    x = (uint8 *)malloc(t * sizeof(*x));
    y = (uint8 *)malloc(t * sizeof(*y));
```

```
f = (FILE **)malloc(t * sizeof(*f));
if (x == NULL || y == NULL || f == NULL)
    return -10;
i = 0;
// Считываем все файлы из директории dirname
while ((entry = readdir(dir)) != NULL && i < t)
{
    // Ищем в директории те файлы, которые содержат название "fraction"
    if (!strncmp(FRACTION_NAME, entry->d_name, strlen(FRACTION_NAME)))
    {
        strcpy(name + len, entry->d_name);
        // Открываем файлы-доли для чтения
        if ((f[i] = fopen(name, "rb")) == NULL)
            return 20;
        // Проверяем, что все файлы-доли имеют одинаковый размер
        if (i == 0)
            size = Size(name);
        else if(size != Size(name))
            return 30;
        i++;
    }
}
// если файлов-долей секрета < t, то секрет нельзя восстановить
if (i < t)
    return 40;

// Открываем файл-секрет для записи
if ((fsecret = fopen(fname, "wb")) == NULL)
    return 10;
// Считываем номера собравшихся участников в массив x
for(i = 0; i < t; i++)
    fread(&x[i], sizeof(x[i]), 1, f[i]);

// Вычисляем, сколько целых буферов обмена помещается в файл
q = (size - sizeof(uint8)) / BUFSIZE;
r = size - sizeof(uint8) - q * BUFSIZE;

// Считываем доли из файлов-долей и восстанавливаем файл-секрет
for(j = 0; j < q; j++)
{
    for(i = 0; i < t; i++)
```

```
            fread(bufy[i], sizeof(uint8), BUFSIZE, f[i]);
        for(k = 0; k < BUFSIZE; k++)
        {
            for(i = 0; i < t; i++)
                y[i] = bufy[i][k];
            bufs[k] = Restoring_secret(x, y, t);
        }
        fwrite(bufs, sizeof(*bufs), BUFSIZE, fsecret);
    }
    if (r > 0)
    {
        for(i = 0; i < t; i++)
            fread(bufy[i], sizeof(uint8), r, f[i]);
        for(k = 0; k < r; k++)
        {
            for(i = 0; i < t; i++)
                y[i] = bufy[i][k];
            bufs[k] = Restoring_secret(x, y, t);
        }
        fwrite(bufs, sizeof(*bufs), r, fsecret);
    }

    closedir(dir);
    fclose(fsecret);
    for(i = 0; i < t; i++)
        fclose(f[i]);
    free(x); free(y); free(f); free(bufs); free(bufy);
    return 0;
}

int main()
{
    long n, t;
    long i;
    char dirname[] = "d:\\fractions_of_a_secret"; // директория для файлов-долей
    char fsecret_name[] = "d:\\a.pdf"; // файл-секрет
    char fsecret_name2[] = "d:\\b.pdf"; // файл-секрет после восстановления

    srand(time(NULL));
    // Вычисление степеней образующего элемента alpha и
    // построение таблицы дискретных логарифмов
```

```
    deg_alpha[0] = 1;
    for (i = 1; i <= NMAX; i++)
    {
        deg_alpha[i] = Product_alpha(deg_alpha[i-1]);
        deg_alpha[NMAX + i] = deg_alpha[i]; // чтобы не брать "% NMAX"
        log_alpha[deg_alpha[i]] = i;
    }

    n = 10;  // задаем число участников n <= 255, так как |GF(2^8)|=256
    t = 7;  // задаем порог t <= n
    // Разделяем файл-секрет на файлы-доли
    printf("Secret_sharing_rezult = %d\n",
            Secret_sharing_file(fsecret_name, dirname, n, t));

    // Перед вызовом следующей функции Restoring_secret_file
    // можно/нужно закомментировать вызов предыдущей функции и
    // в директории dirname можно удалить файлы-доли, оставив
    // не менее t любых файлов-долей для восстановления файла-секрета.
    // Или скопировать в другую директорию не менее t файлов-долей,
    // записав при этом в dirname полное имя новой директории
    printf("Restoring_secret_rezult = %d\n",
            Restoring_secret_file(fsecret_name2, dirname, t));

    return 0;
}
```

При первом запуске программы можно закомментировать инструкцию

```
    printf("Restoring_secret_rezult = %d\n",
            Restoring_secret_file(fsecret_name2, dirname, t));
```

В этом случае будет произведено разделение файла-секрета на файлы-доли, которые будут записаны в директорию dirname. Например, после запуска программы с указанными параметрами $n = 10$, $t = 7$ будет создана директория "d:\fractions_of_a_secret", в которую будут записаны файлы fraction1, fraction2,..., fraction10. Заметим, что при успешном разделении файла-секрета на экране должно быть напечатано значение 0.

Если нужно восстановить секрет по произвольным не менее t файлам-долям, то можно закомментировать инструкцию

```
    printf("Secret_sharing_rezult = %d\n",
            Secret_sharing_file(fsecret_name, dirname, n, t));
```

В этом случае по файлам-долям, которые находятся в директории dirname, файл-секрет будет восстановлен. Например, для рассматриваемого случая $n = 10$, $t = 7$ в директории

"d:\fractions_of_a_secret" можно оставить не менее t файлов-долей и запустить программу. Тогда файл-секрет будет восстановлен.

3. Программная реализация СРС на основе равновесных двоичных кодов

Данная схема интересна тем, что восстановление секрета по долям проходит очень просто, но платой за эту простоту являются размеры долей. При этом при небольшом числе участников и небольшого объема секрета восстановление этого секрета возможно даже без использования ПК. Поэтому данная схема разделения секрета тоже заслуживает внимания.

Опишем эту схему. Пусть V_n — множество всех двоичных векторов длины n. Пусть $n, h \in \mathbb{N}$, $1 \le h \le n$. В множестве V_n рассмотрим подмножество $R(n,h)$, состоящее из всех таких элементов множества V_n, которые имеют в своем составе ровно h единиц (т. е. вес каждого такого вектора равен h). Очевидно, что $|R(n,h)| = C_n^h$. Расположив все элементы множества $R(n,h)$ по столбцам, получим матрицу A порядка $n \times C_n^h$. Например, пусть $n = 5$, $h = 3$. Тогда

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

Для матрицы A верны следующие свойства. Пусть $1 \le t \le n$ и $h = n-t+1$. Выберем в матрице A произвольные t строк и составим из них матрицу B. Тогда каждый столбец матрицы B содержит хотя бы одну единицу. Если в матрице A выбрать произвольные $t-1$ строк и составить из них матрицу C, то в матрице C найдется хотя бы один столбец, состоящий только из нулей.

Пусть s — некоторый секрет. «Нарежем» его на $m = C_n^{n-t+1}$ частей $k_1,\ldots,k_m$. Например, если $s \in G$, где G — некоторая аддитивная абелева группа, то можно поступить следующим образом. Сгенерируем некоторым случайным образом элементы $k_1, \ldots, k_{m-1} \in G$ и определим $k_m = s - k_1 - \ldots - k_{m-1}$. Тогда секрет s делится на m частей $k_1, \ldots, k_m$ и только наличие всех m частей позволяет восстановить секрет s. В качестве группы G в нашем случае будет выступать аддитивная группа поля $GF(2^8)$ (иными словами, множество V_8 с поразрядной операцией сложения по модулю два).

Теперь j-й столбец матрицы A умножим на k_j, $j = 0, 1, \ldots, m-1$ (нумеровать будем с нуля). Остается каждому участнику передать строку итоговой матрицы A, номер которой соответствует номеру участника.

Понятно, что любые t участников могут собрать вместе все части $k_1, \ldots, k_m$ секрета, но никакие $t-1$ участников собрать весь набор $k_1, \ldots, k_m$ не смогут. При этом если используемая (m,m)-схема является совершенной, то и сама схема на основе равновесных кодов будет являться совершенной.

В нашей программе, в частности, будут присутствовать следующие функции.

Логическая функция

```
int n_choose_k(long n, long k, long *choose)
```

вычисляет C_n^k — число сочетаний из n по k и вычисленное значение записывает в $*choose$. Вычисления проводятся на основе треугольника Паскаля, чтобы не было переполнения типа при использовании умножения. Здесь принципиально переменная *choose* имеет тип long, а не 64-разрядный целочисленный тип, так как если на некотором шаге промежуточное или итоговое значение C_n^k превысит значения, которое записано в константу BOUND, то это значит, что доли секрета будут занимать очень много места и могут не поместиться в памяти компьютера, поэтому вычисление будет завершено, а функция возвратит ложное значение. Если итоговое значение не превосходит BOUND, то функция вернет истинное значение. Например, пусть $n = 40$, $t = 21$. Тогда $h = 20$, $m = C_{40}^{20} = 137846528820$, поэтому в этом случае размер доли секрета каждого участника будет занимать более 100 гигабайт и не поместится в оперативной памяти компьютера. Для параметров $n = 40$, $t = 21$ и даже более того хорошо подойдет схема Шамира, а схема на основе равновесных кодов хороша простым алгоритмом восстановления секрета, поэтому может пригодиться для небольших n, в частности, для ручного восстановления секрета.

Функция

```
int Init_A(uint8 **A, long n, long m, long h)
```

инициализирует упомянутую выше матрицу A размером $n \times m$, где n — число участников разделения секрета, $m = C_n^h$ — длина доли секрета. Для заполнения матрицы A, каждый столбец которого является двоичным вектором длины n веса h, используется вспомогательный массив C, в который будут записываться всевозможные сочетания из n-элементного множества $\{0, 1, \ldots, n-1\}$ по h. Останется в соответствии с очередным сгенерированным сочетанием на соответствующие позиции в очередном столбце матрицы A поставить единицы.

Функция

```
int Secret_sharing(uint8 s, uint8 **A, uint8 **y,  long n, long m)
```

на основе секрета s и матрицы A в матрицу долей y записывает доли секрета. Матрицы A и y имеют размер $n \times m$.

Функция

```
uint8 Restoring_secret(uint8 **y, long t, long m)
```

восстанавливает значение секрета по матрице долей y размером $t \times m$, что означает, что собрались вместе не менее t участников и предоставили свои доли секрета.

Функция

```
int Secret_sharing_file(char *fname, char *dirname, long n, long t)
```

разделяет файл-секрет с именем fname на n участников с порогом t и файлы-доли секрета записывает в директорию с именем dirname.

Функция

```
int Restoring_secret_file(char *fname, char *dirname, long n, long t)
```

пробегает по директории с именем dirname, отыскивает в ней файлы-доли не менее t участников и в случае успешного нахождения не менее t файлов-долей восстанавливает файл-секрет, который записывает в файл с именем fname.

Сама программа примет следующий вид.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <dirent.h>
#include <dir.h>
#include <sys/types.h>

#define LEN 1024 // максимальная длина строки текстового файла
#define BUFSIZE 4048 // размер буфера обмена

#define FRACTION_NAME "fraction" // имена файлов долей секрета
#define FILE_LEN 512 // максимальная длина полного имени файла

#define BOUND 100000000 // верхняя граница числа столбцов матрицы A

typedef unsigned char uint8;
typedef unsigned short uint16;
typedef unsigned long uint32;
typedef unsigned long long uint64;

// Размер файла в байтах
uint64 Size(const char *fileName)
{
   FILE *f;
   uint64 size;
   if ((f = fopen(fileName, "rb")) == NULL)
      return 0;
   fseek(f, 0, SEEK_END);
   size = ftell(f);
   fclose(f);
```

```
   return size;
}

// Вычисление числа сочетаний из n элементов по k
// с помощью треугольника Паскаля, которое записывается
// в *choose. Если это значение > BOUND, то функция возвращает ложь
int n_choose_k(long n, long k, long *choose)
{
    uint64 *prev, // предыдущая строка треугольника Паскаля
           *cur, // текущая строка треугольника Паскаля
    *a, *b, *buf;
    uint8 i, j;
    int flag = 1;
    if (k > n)
    {
         *choose = 0;
         return 1;
    }
    if (k == 0 || k == n)
    {
         *choose = 1;
         return 1;
    }
    // Выделяем место в памяти для динамических массивов
    a = (uint64 *)calloc((n+1), sizeof(*a));
    b = (uint64 *)calloc((n+1), sizeof(*b));
    if (a == NULL || b == NULL)
        return 0;
    prev = a; cur = b;
    prev[0] = 1;
    for(i = 1; i <= n && flag; i++)
    {
        cur[0] = 1;
        for(j = 1; j <= i && j <= k && flag; j++)
        {
            cur[j] = prev[j-1] + prev[j];
            if (cur[j] > BOUND)
                flag = 0;
        }
        buf = prev; prev = cur; cur = buf;
    }
```

```
    *choose = prev[k];
    free(a); free(b);
    return flag;
}


// Инициализация матрицы A размером n на m, каждый столбец которой
// имеет вес h
int Init_A(uint8 **A, long n, long m, long h)
{
    long i, j, k = 0;
    uint8 *C; // массив, содержащий сочетания элементов из n по h

    for(i = 0; i < n; i++)
        for(j = 0; j < m; j++)
            A[i][j] = 0;
    C = (uint8 *)malloc(h*sizeof(*C));
    if (C == NULL)
        return 0;
    for(i = 0; i < h; i++)
        C[i] = i;
    i = h - 1;
    while(i >= 0)
    {
        // Ставим в k-м столбце матрицы A единицы на позиции,
        // номера которых содержатся в массиве C, где C - сочетания
        for(j = 0; j < h; j++)
            A[C[j]][k] = 1;
        k++;
        // Генерируем следующее сочетание
        if (C[h - 1] == n - 1)
            i--;
        else i = h - 1;
        if (i >= 0)
            for(j = h - 1; j >= i; j--)
                C[j] = C[i] + j - i + 1;
    }
    free(C);
    return 1;
}
```

```
// Выделение памяти для двумерного динамического массива
// размером n на m для файлов-долей секрета
uint8 **Allocate(long n, long m)
{
    uint8 **a;
    long i;
    a = (uint8 **)malloc(n*m*sizeof(uint8) + n*sizeof(uint8 *));
    if (a != NULL)
        for(i = 0; i < n; i++)
             a[i] = (uint8 *)(a + n) + i*m;
    return a;
}

// Разделение секрета s с порогом t в соответствии с матрицей A
// размером n на m, которые записываются в матрицу y
int Secret_sharing(uint8 s, uint8 **A, uint8 **y,  long n, long m)
{
    uint8 *a; // доли (m,m) - вспомогательной схемы
    uint32 i, j;
    a = (uint8 *)malloc(m * sizeof(*a));
    if (a == NULL)
        return 0;
    // Генерация долей (m,m) - пороговой схемы
    a[m - 1] = s;
    // Здесь для примера используется небезопасный генератор rand()
    for(i = 0; i < m - 1; i++)
    {
        a[i] = rand() % 256;
        a[m - 1] ^= a[i];
    }
    // В итоге, a[n-1] = s - (a[0]+...+a[n-2])
    // Вычисление долей секрета s и запись их в матрицу y
    for(i = 0; i < n; i++)
        for(j = 0; j < m; j++)
            y[i][j] = (A[i][j] ? a[j] : 0);
    free(a);
    return 1;
}

// Восстановление секрета s по t долям,
// m отвечает за (m,m) - пороговую схему
```

```
uint8 Restoring_secret(uint8 **y, long t, long m)
{
    long i, j;
    uint8 s; // секрет
    // flag отвечает, чтобы каждая доля (m,m) - пороговой схемы
    // была использована ровно один раз
    int flag;
    s = 0;
    for(j = 0; j < m; j++)
    {
        flag = 1;
        for(i = 0; i < t && flag; i++)
            if (y[i][j])
            {
                s ^= y[i][j];
                flag = 0;
            }
    }
    return s;
}

// Разделение файла-секрета fname на n участников с порогом t.
// Файлы-доли секрета будут помещены в директорию dirname
int Secret_sharing_file(char *fname, char *dirname, long n, long t)
{
    FILE *fsecret, // файл-секрет
        **f; // файлы-доли секрета
    char name[FILE_LEN] = {0}, st[FILE_LEN];
    long h, i, j, len, count, m;
    uint8 s; // секрет
    uint8 *bufs; // буфер обмена для файла-секрета
    uint8 **y; // доли секрета
    uint8 **A; // матрица, каждый столбец которой имеет вес h

    h = n - t + 1; // h - вспомогательный параметр, число единиц в векторах
    if (!n_choose_k(n, h, &m))
        return -5;
    // Выделяем место в памяти для буфера обмена
    bufs = (uint8 *)malloc(BUFSIZE*sizeof(*bufs));
    // Выделяем место в памяти для долей секрета
    A = Allocate(n, m);
```

```
y = Allocate(n, m);
if (bufs == NULL || A == NULL || y == NULL)
    return 5;
 // Инициализируем матрицу A, в которой каждый столбец имеет вес h
Init_A(A, n, m, h);
// Создаем директорию для файлов-долей
mkdir(dirname);
// Открываем файл-секрет для чтения
if ((fsecret = fopen(fname, "rb")) == NULL)
    return 10;
// В переменную name будут по очереди записаны полные имена файлов-долей
// секрета. Изначально в строковую переменную  name записываем полное
// название директории dirname и добавляем в конец "\fraction".
strcpy(name, dirname);
strcat(name, "\\");
strcat(name, FRACTION_NAME);
len = strlen(name);
// Динамический массив файлов-указателей-долей секрета
f = (FILE **)malloc(n * sizeof(*f));
if (f == NULL)
    return 20;
// Открываем файлы-доли секрета для записи
for(i = 0; i < n; i++)
{
    itoa(i, st, 10); // переводим номер участника в строку st
    strcpy(name + len, st); // добавляем номер к названию файла-доли секрета
    f[i] = fopen(name, "wb"); // открываем файл-долю для записи
    if (f[i] == NULL)
        return 30;
}
// Разделяем файл-секрет на доли, которые записываем в файлы-долей,
// используя для ускорения работы буфер обмена bufs
while (count = fread(bufs, sizeof(*bufs), BUFSIZE, fsecret))
{
    for(j = 0; j < count; j++)
    {
        // Вычисляем доли y секрета bufs[j]
        Secret_sharing(bufs[j], A, y, n, m);
        // Записываем доли секрета в файлы-доли секрета
        for(i = 0; i < n; i++)
           fwrite(y[i], sizeof(uint8), m, f[i]);
```

```
        }
    }

    fclose(fsecret);
    for(i = 0; i < n; i++)
        fclose(f[i]);
    free(f); free(y); free(A); free(bufs);
    return 0;
}

// Восстановление файла-секрета по t файлам-долей
int Restoring_secret_file(char *fname, char *dirname, long n, long t)
{
    FILE *fsecret, //файл-секрет
        **f; // файлы-доли секрета
    struct dirent *entry; // для "вытаскивания" файлов-долей секрета из директории
    DIR *dir; // указатель-директория
    char name[FILE_LEN] = {0}, st[FILE_LEN];
    long i, len, h, m;
    uint8 s;
    uint64 j, k, size, q, r;
    uint8 *bufs; // буфер обмена для файла-секрета
    uint8 **y; // доли секрета

    h = n - t + 1; // h - вспомогательный параметр, число единиц в векторах
    if (!n_choose_k(n, h, &m))
        return -5;
    // Выделяем место в памяти для буфера обмена
    bufs = (uint8 *)malloc(BUFSIZE*sizeof(*bufs));
    // Выделяем место в памяти для долей секрета
    y = Allocate(t, m);
    if (bufs == NULL || y == NULL)
        return 5;

    // Открываем директорию, откуда будем брать файлы-доли секрета
    if ((dir = opendir(dirname)) == NULL)
        return 5;
    strcpy(name, dirname);
    strcat(name, "\\");
    len = strlen(name);
    // Создаем динамические массивы:
```

```
// f - файловые указатели-доли секрета
f = (FILE **)malloc(t * sizeof(*f));
if (f == NULL)
    return -10;
i = 0;
// Считываем все файлы из директории dirname
while ((entry = readdir(dir)) != NULL && i < t)
{
    // Ищем в директории те файлы, которые содержат название "fraction"
    if (!strncmp(FRACTION_NAME, entry->d_name, strlen(FRACTION_NAME)))
    {
        strcpy(name + len, entry->d_name);
        // Открываем файлы-доли для чтения
        if ((f[i] = fopen(name, "rb")) == NULL)
            return 20;
        // Проверяем, что все файлы-доли имеют одинаковый размер
        if (i == 0)
            size = Size(name);
        else if(size != Size(name))
            return 30;
        i++;
    }
}
// если файлов-долей секрета < t, то секрет нельзя восстановить
if (i < t)
    return 40;

// Открываем файл-секрет для записи
if ((fsecret = fopen(fname, "wb")) == NULL)
    return 10;

// Вычисляем, сколько целых блоков длины m помещается в файл
q = size / (m * sizeof(uint8));
r = size - q * (m * sizeof(uint8));
if (r > 0)
    return 15;

k = 0;
// Считываем доли из файлов-долей и восстанавливаем файл-секрет
for(j = 0; j < q; j++)
{
```

```
        for(i = 0; i < t; i++)
            fread(y[i], sizeof(uint8), m, f[i]);
        s = Restoring_secret(y, t, m);
        bufs[k++] = s;
        if (k == BUFSIZE)
        {
            fwrite(bufs, sizeof(*bufs), BUFSIZE, fsecret);
            k = 0;
        }
    }
    if (k > 0)
        fwrite(bufs, sizeof(*bufs), k, fsecret);

    closedir(dir);
    fclose(fsecret);
    for(i = 0; i < t; i++)
        fclose(f[i]);
    free(y); free(f); free(bufs);
    return 0;
}

// Двоичное представление целого числа a записывается в строку s
void Itoa(uint8 a, char *s)
{
    int i, j, bit;
    for (i = sizeof(a)*8 - 1, j = 0; i >= 0; i--, j++)
    {
       bit = (a >> i) & 1;
       s[j] = '0' + bit;
    }
    s[j] = '\0';
}

// Вывод содержимого "небольшого" файла на экран.
// m имеет прежний смысл - длина доли.
int Print_file(char *fname, long m)
{
    FILE *f;
    uint8 *a;
    long i;
    char s[10];
```

```
    if ((f = fopen(fname, "rb")) == NULL)
        return 1;
    a = (uint8 *)malloc(m * sizeof(*a));
    if (a == NULL)
        return 2;
    while(fread(a, sizeof(*a), m, f))
    {
        for(i = 0; i < m; i++)
        {
            Itoa(a[i], s);
            printf(s);
            printf(" ");
        }
        puts("");
    }

    fclose(f);
    free(a);
    return 0;
}

int main()
{
    char dirname[] = "d:\\fractions_of_a_secret"; // директория для файлов-долей
    char fsecret_name[] = "d:\\a.data"; // файл-секрет
    char fsecret_name2[] = "d:\\b.data"; // файл-секрет после восстановления
    long n, t, m;

    srand(time(NULL));

    n = 7;
    t = 3;
    // Разделяем файл-секрет на файлы-доли
    printf("Secret_sharing_rezult = %d\n",
            Secret_sharing_file(fsecret_name, dirname, n, t));
    // Восстанавливаем файл-секрет по файлам-долям
    printf("Restoring_secret_rezult = %d\n",
            Restoring_secret_file(fsecret_name2, dirname, n, t));
    return 0;
}
```

При первом запуске программы можно закомментировать инструкцию

```
printf("Restoring_secret_rezult = %d\n",
        Restoring_secret_file(fsecret_name2, dirname, n, t));
```

В этом случае будет произведено разделение файла-секрета на файлы-доли, которые будут записаны в директорию dirname. Например, после запуска программы с указанными параметрами $n = 7$, $t = 3$ будет создана директория "d:\fractions_of_a_secret", в которую будут записаны файлы fraction0, fraction1,..., fraction6. Заметим, что при успешном разделении файла-секрета на экране должен быть напечатан 0.

Если нужно восстановить секрет по произвольным не менее t файлам-долям, то можно закомментировать инструкцию

```
printf("Secret_sharing_rezult = %d\n",
        Secret_sharing_file(fsecret_name, dirname, n, t));
```

В этом случае по файлам-долям, которые находятся в директории dirname, файл-секрет будет восстановлен. Например, для рассматриваемого случая $n = 7$, $t = 3$ в директории "d:\fractions_of_a_secret" можно оставить не менее t файлов-долей и запустить программу. Тогда файл-секрет будет восстановлен.

Но самое важное в этой схеме не программный способ восстановления секрета (для этого имеется совершенная идеальная схема Шамира), а возможность это сделать вручную (когда под рукой ничего нет, только запись на бумаге со значением доли секрета), так как для восстановления секрета не нужно проделывать каких-то сложных действий. Здесь как раз понадобится функция Print_file. Предположим, что секрет небольшой и участников немного. Каждый участник может запустить программу, передав в качестве параметра функции Print_file имя своего файла-доли секрета. Эта функция выведет на экран (можно сделать и вывод в текстовый файл) доли секрета в виде двоичного вектора. Участники могут записать каждый свою долю секрета, например, на бумаге. Например, пусть $n = 4$, $t = 3$. В этом случае $m = C_4^2 = 6$. Предположим, что каждый участник записал себе следующие данные.

1-й участник:

00100011 10010111 11101000 00000000 00000000 00000000
00111110 01101000 11111001 00000000 00000000 00000000
01001010 01000101 10001110 00000000 00000000 00000000
00111101 00100010 01001000 00000000 00000000 00000000
11111110 01100000 10110101 00000000 00000000 00000000

2-й участник:

00100011 00000000 00000000 10011101 01110011 00000000
00111110 00000000 00000000 00001001 11000010 00000000
01001010 00000000 00000000 00110101 00101010 00000000
00111101 00000000 00000000 11000011 00111101 00000000
11111110 00000000 00000000 11111000 01011000 00000000

3-й участник:

00000000 10010111 00000000 10011101 00000000 10110011
00000000 01101000 00000000 00001001 00000000 01100110
00000000 01000101 00000000 00110101 00000000 10011101
00000000 00100010 00000000 11000011 00000000 10101101
00000000 01100000 00000000 11111000 00000000 10001110

4-й участник:

00000000 00000000 11101000 00000000 01110011 10110011
00000000 00000000 11111001 00000000 11000010 01100110
00000000 00000000 10001110 00000000 00101010 10011101
00000000 00000000 01001000 00000000 00111101 10101101
00000000 00000000 10110101 00000000 01011000 10001110

У каждого участника 5 строк, значит файл-секрет занимает 5 байт. Каждая строка — это доля секрета для восстановления очередного байта. Предположим, что собрались первые три участника. Восстановим первый байт файла-секрета. Нужно собрать 6 долей вспомогательной (m, m)-схемы. Каждый участник смотрит на первую строку своих записей. У троих собравшихся участников имеются значения

00100011, 10010111, 11101000, 10011101, 01110011, 10110011.

После поразрядного складывания этих векторов по модулю два получаем значение 00000001=1.

Теперь собравшиеся три участника смотрят на свои вторые строки. Они собирают значения

00111110, 01101000, 11111001, 00001001, 11000010, 01100110.

По ним они получают значение 00000010=2. Аналогичным образом они восстанавливают значения оставшихся трех байт и в итоге восстанавливают секрет 12345. Как видно, никаких вычислительных устройств в этом случае не понадобилось. Заметим также, что значения долей можно хранить и в другом формате, например, шестнадцатеричном, тогда записи будут более компактными. Например, для первого участника доли можно записать следующим образом:

23	97	$e8$	00	00	00
$3e$	68	$f9$	00	00	00
$4a$	45	$8e$	00	00	00
$3d$	22	48	00	00	00
fe	60	$b5$	00	00	00

4. Программная реализация схемы Ито—Саито—Нишизеки

В отличие от рассмотренной выше схемы Шимира в данном случае не требуется строить схему над полем, здесь достаточно определить некоторую аддитивную абелеву группу. В качестве такой группы будет выступать аддитивная группа поля $GF(2^8)$ (иными словами,

множество двоичных длины 8 с поразрядной операцией сложения по модулю два). В работе [8] приводится программная реализация построения множества правомочных коалиций, множества неправомочных коалиций, множества минимальных правомочных коалиций, множества максимальных неправомочных коалиций для структур доступа для схем разделения секрета, если хотя бы одно их перечисленных множеств известно. Также строится кумулятивный массив для схемы Ито—Саито—Нишизеки. Дополним ту программу, чтобы была возможность работать с файлами-секретами.

Добавим библиотечные файлы:

```
#include <string.h>
#include <time.h>
#include <dirent.h>
#include <dir.h>
#include <sys/types.h>
```

Добавим константы и средство typedef:

```
#define LEN 1024 // максимальная длина строки текстового файла
#define BUFSIZE 4048 // размер буфера обмена

#define FRACTION_NAME "fraction" // имена файлов долей секрета
#define FILE_LEN 512 // максимальная длина полного имени файла

typedef unsigned short uint16;
typedef unsigned long long uint64;
```

Полностью удалим функцию main (вместе с содержимым) из представленной в работе [8] программы, чтобы заменить ее на представленную ниже для возможности работы с файлами.

После удаления функции main добавим снизу к функциям из работы [8] следующие функции.

```
// Размер файла в байтах
uint64 Size(const char *fileName)
{
   FILE *f;
   uint64 size;
   if ((f = fopen(fileName, "rb")) == NULL)
      return 0;
   fseek(f, 0, SEEK_END);
   size = ftell(f);
   fclose(f);
   return size;
}
```

Данная функция возвращает размер файла в байтах.

```
// Перевод строки s с номерами участников, входящих в коалицию,
// в переменную a типа uint32
int Coalition_to_uint32(char *s, uint32 *a)
{
    uint32 i, j;
    int flag = 0; // flag отвечает за то, не были ли считаны пустые строки
    char *number;
    (*a) = 0;
    for(number = strtok(s, " ,.\t\n"); number != NULL;
        number = strtok(NULL, " ,.\t\n"))
    {
        j = atoi(number);
        (*a) |= (1 << j);
        flag = 1;
    }
    return flag;
}
```

Данная функция переводит строковое представление коалиции участников в числовое представление (более точно, в двоичную последовательность). Например, коалиция участников 0, 1, 5 при $N = 7$ (которую можно задать через пробелы 0 1 5) в двоичное представление 0100011. В шестнадцатеричном виде эта коалиция примет вид 0x23. Заметим, что в этой функции используется функция strtok. Более эффективный метод разбиения строки на лексемы можно найти в работе [9]. Так как множества коалиций (например, множество минимальных правомочных коалиций R_min) удобно задавать в виде текстового файла, в котором каждая стока содержит номера участников очередной коалиции, записанных через запятые или через пробел, то после считывания строк из такого файла нужно эти строки преобразовывать в числовой вид (двоичную последовательность).

```
// Инициализация множества коалиций A из текстового файла, каждая строка которого
// является коалицией участников, записанных через пробел,
// серию пробелов или запятые
int Init_set_of_coalitions(const char *fname, uint32 *A, uint32 *na)
{
    FILE *f;
    char s[LEN];
    uint32 a;

    if ((f = fopen(fname, "r")) == NULL)
        return 1;
```

```
    *na = 0;
    while(fgets(s, LEN, f) != NULL)
        // Выделяем из строки номера участников и формируем этим самым коалицию a
        if (Coalition_to_uint32(s, &a))
            A[(*na)++] = a;

    fclose(f);
    return 0;
}
```

Данная функция считывает строки из текстового файла, в котором каждая строка — коалиция участников, записанных в виде номеров через пробелы или запятые (см. описание предыдущей функции). При считывании очередной строки происходит вызов функции Coalition_to_uint32, которая переводит строковое изображение коалиции в число. После считывания всех строк текстового файла в множество A будут записаны все коалиции, представленные в файле с именем fname.

```
// Разделение секрета s на N долей в соответствии с кумулятивным массивом C
// размером N на m, которые записываются в матрицу y
int Secret_sharing(uint8 s, uint8 **C, uint8 **y, uint32 m)
{
    uint8 *a; // доли (m,m) - вспомогательной схемы
    uint32 i, j;
    a = (uint8 *)malloc(m * sizeof(*a));
    if (a == NULL)
        return 0;
    // Генерация долей (m,m) - пороговой схемы
    a[m - 1] = s;
    // Здесь для примера используется небезопасный генератор rand()
    for(i = 0; i < m - 1; i++)
    {
        a[i] = rand() % 256;
        a[m - 1] ^= a[i];
    }
    // В итоге, a[n-1] = s - (a[0]+...+a[n-2])
    // Вычисление долей секрета s и запись их в матрицу y
    for(i = 0; i < N; i++)
        for(j = 0; j < m; j++)
            y[i][j] = (C[i][j] ? a[j] : 0);
    free(a);
    return 1;
}
```

Данная функция получает в качестве параметра секрет s и на основе заранее построенного кумулятивного массива C записывает доли секрета s в двумерный динамический массив y размером $N \times m$. Каждому участнику позднее достанется строка массива y, номер которой совпадает с номером участника.

```
// Восстановление секрета s по долям правомочной коалиции,
// состоящей из n участников. m отвечает за (m,m) - пороговую схему
uint8 Restoring_secret(uint8 **y, uint32 n, uint32 m)
{
    uint32 i, j;
    uint8 s; // секрет
    // flag отвечает, чтобы каждая доля (m,m) - пороговой схемы
    // была использована ровно один раз
    int flag;
    s = 0;
    for(j = 0; j < m; j++)
    {
        flag = 1;
        for(i = 0; i < n && flag; i++)
            if (y[i][j])
            {
                s ^= y[i][j];
                flag = 0;
            }
    }
    return s;
}
```

Данная функция восстанавливает секрет s по двумерному массиву долей y. Для этого достаточно в каждом столбце массива y выбрать по одному ненулевому элементу (при их наличии) и просуммировать их.

```
// Разделение файла-секрета fname на N участников на основе
// кумулятивного массива C размером N на m.
// Файлы-доли секрета будут помещены в директорию dirname.
int Secret_sharing_file(char *fname, char *dirname, uint8 **C, uint32 m)
{
    FILE *fsecret, // файл-секрет
        **f; // файлы-доли секрета
    char name[FILE_LEN] = {0}, st[FILE_LEN];
    uint32 i, j, len, count;
    uint8 s; // секрет
```

```
uint8 *bufs; // буфер обмена для файла-секрета
uint8 **y; // доли секрета

// Выделяем место в памяти для буфера обмена
bufs = (uint8 *)malloc(BUFSIZE*sizeof(*bufs));
// Выделяем место в памяти для долей секрета
y = Allocate(m);
if (bufs == NULL || y == NULL)
    return 5;
// Создаем директорию для файлов-долей
mkdir(dirname);
// Открываем файл-секрет для чтения
if ((fsecret = fopen(fname, "rb")) == NULL)
    return 10;
// В переменной name будут по-очереди записаны полные имена файлов-долей секрета.
// Изначально в строковую переменную  name записываем полное название
// директории dirname и добавляем в конец "\fraction".
strcpy(name, dirname);
strcat(name, "\\");
strcat(name, FRACTION_NAME);
len = strlen(name);
// Динамический массив файлов-указателей-долей секрета
f = (FILE **)malloc(N * sizeof(*f));
if (f == NULL)
    return 20;
// Открываем файлы-доли секрета для записи
for(i = 0; i < N; i++)
{
    itoa(i, st, 10); // переводим номер участника в строку st
    strcpy(name + len, st); // добавляем номер к названию файла-доли секрета
    f[i] = fopen(name, "wb"); // открываем файл-долю для записи
    if (f[i] == NULL)
        return 30;
}
// Запишем номера участников в начало каждого файла с долями
for(i = 0; i < N; i++)
    fwrite(&i, sizeof(uint8), 1, f[i]);
// Разделяем файл-секрет на доли, которые записываем в файлы-долей,
// используя для ускорения работы буфер обмена bufs
while (count = fread(bufs, sizeof(*bufs), BUFSIZE, fsecret))
{
```

```
        for(j = 0; j < count; j++)
        {
            // Вычисляем доли y секрета bufs[j]
            Secret_sharing(bufs[j], C, y, m);
            // Записываем доли секрета в файлы-доли секрета
            for(i = 0; i < N; i++)
               fwrite(y[i], sizeof(uint8), m, f[i]);
        }
    }

    fclose(fsecret);
    for(i = 0; i < N; i++)
        fclose(f[i]);
    free(f); free(y); free(bufs);
    return 0;
}
```

Данная функция производит разделение файла-секрета с именем fname на N участников. Для ускорения работы программы здесь используется буфер обмена bufs для работы с файлами. С помощью инструкции mkdir(dirname) создается директория с именем dirname (если она уже присутствует, то ничего не происходит). Далее происходит считывание целыми кусками размера BUFSIZE байтов из файла-секрета, разделение каждого из этих байт на доли секрета, которые записываются в файлы-доли. Доли секрета записываются в файлы-доли с помощью инструкции fwrite(y[i], sizeof(uint8), m, f[i]). Если число m достаточно большое, то такие записывания будут достаточно быстрыми. Если число m маленькое, то здесь можно тоже использовать буфер обмена для ускорения работы с файлами. Оставим этот момент для самостоятельной работы желающим реализовать очень эффективную реализацию схемы Ито—Саито—Нишизеки.

```
// Восстановление файла-секрета по файлам-долям,
// которые находятся в директории dirname
int Restoring_secret_file(char *fname, char *dirname, uint32 m, uint32 *R, uint32 nr)
{
    FILE *fsecret, //файл-секрет
        **f; // файлы-доли секрета
    struct dirent *entry; // для "вытаскивания" файлов-долей секрета из директории
    DIR *dir; // указатель-директория
    char name[FILE_LEN] = {0}, st[FILE_LEN];
    uint32 i, len;
    uint32 count; // число участников собравшейся коалиции
    uint32 x; // коалиция собравшихся участника
    uint8 a;
```

```
uint8 s;
uint64 j, k, size, q, r;
uint8 *bufs; // буфер обмена для файла-секрета
uint8 **y; // доли секрета

// Выделяем место в памяти для буфера обмена
bufs = (uint8 *)malloc(BUFSIZE * sizeof(*bufs));
// Выделяем место в памяти для долей секрета
y = Allocate(m);
if (bufs == NULL || y == NULL)
    return 2;

// Открываем директорию, откуда будем брать файлы-доли секрета
if ((dir = opendir(dirname)) == NULL)
    return 5;
strcpy(name, dirname);
strcat(name, "\\");
len = strlen(name);
// Создаем динамические массивы:
// f - файловые указатели-доли секрета
f = (FILE **)malloc(N * sizeof(*f));
if (f == NULL)
    return -10;
i = 0;
// Считываем все файлы из директории dirname
while ((entry = readdir(dir)) != NULL)
{
    // Ищем в директории те файлы, которые содержат название "fraction"
    if (!strncmp(FRACTION_NAME, entry->d_name, strlen(FRACTION_NAME)))
    {
        strcpy(name + len, entry->d_name);
        // Открываем файлы-доли для чтения
        if ((f[i] = fopen(name, "rb")) == NULL)
            return 20;
        // Проверяем, что все файлы-доли имеют одинаковый размер
        if (i == 0)
            size = Size(name);
        else if(size != Size(name))
            return 30;
        i++;
    }
```

```
}
count = i;

x = 0;
// Считываем номера собравшихся участников и создаем из них коалицию x
for(i = 0; i < count; i++)
{
    fread(&a, sizeof(uint8), 1, f[i]);
    x |= (1 << a);
}
// Проверяем, является ли коалиция x правомочной,
// т. е. принадлежит множеству всех правомочных коалиций R
if (!Binary_search(R, nr, x))
    return 50;

// Открываем файл-секрет для записи
if ((fsecret = fopen(fname, "wb")) == NULL)
    return 10;
// Вычисляем, сколько целых блоков длины m помещается в файл
q = (size - sizeof(uint8)) / (m * sizeof(uint8));
r = size - sizeof(uint8) - q * (m * sizeof(uint8));
if (r > 0)
    return 15;

k = 0;
// Считываем доли из файлов-долей и восстанавливаем файл-секрет
for(j = 0; j < q; j++)
{
    for(i = 0; i < count; i++)
        fread(y[i], sizeof(uint8), m, f[i]);
    s = Restoring_secret(y, count, m);
    bufs[k++] = s;
    if (k == BUFSIZE)
    {
        fwrite(bufs, sizeof(*bufs), BUFSIZE, fsecret);
        k = 0;
    }
}
if (k > 0)
    fwrite(bufs, sizeof(*bufs), k, fsecret);
```

```
    closedir(dir);
    fclose(fsecret);
    for(i = 0; i < N; i++)
        fclose(f[i]);
    free(y); free(f); free(bufs);
    return 0;
}
```

Данная функция считывает содержимое директории с именем dirname и восстанавливает файл-секрет с именем fname. Из начала каждого файла-доли секрета из директории dirname считываются номера участников, после чего происходит проверка с помощью функции бинарного поиска, составляют ли эти участники правомочную коалицию, т. е. принадлежат ли они множеству всех правомочных коалиций R. Если ответ положительный, то происходит восстановление файла-секрета. Заметим, что можно поступить немного иначе. Можно не передавать функции Restoring_secret_file множество всех правомочных коалиций R, а восстановить "файл-секрет". Если коалиция была неправомочной, то за содержимое полученного файла никто ответственность уже не несет. В этом случае нет необходимости заново вычислять множество R перед восстановлением файла-секрета. Но на вопрос, передавать или не передавать указанной функции множество R, пусть ответит для себя читатель (все зависит от конкретных ситуаций).

```
// Вывод коалиций в файл
int PrintFile(char *fname, const uint32 *A, const uint32 m)
{
    FILE *f;
    uint32 i, j;
    if ((f = fopen(fname, "w")) == NULL)
     return 1;
    for(i = 0; i < m; i++)
    {
        for(j = 0; j < N; j++)
            if ((A[i] >> j) & (uint32)1)
                fprintf(f, "%u ", j);
        fputs("\n",f);
    }
    return 0;
    fclose(f);
}
```

Данная функция записывает каждую коалицию из некоторого множества коалиций A в текстовый файл. После вычисления множеств R, Z, R_min, Z_max любую из них (или их все) можно записать в соответствующий текстовый файл построчно, в котором каждая строка

представлена в виде совокупности номеров участников, составляющих коалицию. Например, коалиция 0100011 будет соответствовать строке текстового файла, в котором записаны номера участников, а именно, 0, 1, 5 (или 0 1 5).

Осталось добавить обновленную функцию main.

```
int main()
{
    uint32 nr, // число правомочных коалиций
        nz, // число неправомочных коалиций
        nr_min, // число минимальных правомочных коалиций
        nz_max, // число максимальных неправомочных коалиций
        m;
    // коалиции R и Z будут в лексикографическом порядке
    uint32 *R, // правомочные коалиции
            *Z, // неправомочные коалиции
            *R_min, // минимальные правомочные коалиции
            *Z_max; // максимальные неправомочные коалиции
    // Кумулятивный массив C размера N на m, где
    // N - число участников, m - число максимальных неправомочных коалиций
    uint8 **C;
    uint32 i;
    char dirname[] = "d:\\fractions_of_a_secret"; // директория для файлов-долей
    char fsecret_name[] = "d:\\a.pdf"; // файл-секрет
    char fsecret_name2[] = "d:\\b.pdf"; // файл-секрет после восстановления
    char R_min_name[] = "d:\\R_min.txt"; // файл с минимальными правомочными коалициями

    // Выделяем память для массивов R, Z, R_min, Z_max
    R = (uint32*)malloc(COUNT * sizeof(*R));
    Z = (uint32*)malloc(COUNT * sizeof(*Z));
    R_min = (uint32*)malloc(COUNT * sizeof(*R_min));
    Z_max = (uint32*)malloc(COUNT * sizeof(*Z_max));
    if (R == NULL || Z == NULL || R_min == NULL || Z_max == NULL)
    {
        puts("not enough memory");
        return 1;
    }

    srand(time(NULL));
    // Инициализируем множество R_min на основе содержимого файла R_min_name
    Init_set_of_coalitions(R_min_name, R_min, &nr_min);
```

```
    // Построение R на основе R_min
    R_Rmin(R, &nr, R_min, nr_min);
    // Построение Z на основе R
    Z_R(Z, &nz, R, nr);
    // Построение Z_max на основе Z
    Zmax_Z(Z_max, &nz_max, Z, nz);
    m = nz_max;
    C = Allocate(m); // выделяем память под массив C
    if(C != NULL)
        Init_C(C, Z_max, m); // строим кумулятивный массив C
    else
    {
        puts("not enough memory");
        return 1;
    }

    printf("Secret_sharing_rezult = %d\n",
            Secret_sharing_file(fsecret_name, dirname, C, m));
    printf("Restoring_secret_rezult = %d\n",
            Restoring_secret_file(fsecret_name2, dirname, m, R, nr));

    free(R); free(Z); free(R_min); free(Z_max);
    return 0;
}
```

Перед запуском полученной программы необходимо создать файл R_min_name и записать туда построчно множество минимальных правомочных коалиций. Например, можно построчно записать следующие коалиции:
0, 1, 5
2, 3, 5
2, 4
4, 6
После этого закомментировать инструкцию

```
printf("Restoring_secret_rezult = %d\n",
        Restoring_secret_file(fsecret_name2, dirname, m, R, nr));
```

и запустить программу. В этом случае будет произведено разделение файла-секрета на файлы-доли, которые будут записаны в директорию dirname.

Напомним, что число участников разделения секрета задается в константе N, а общее число непустых коалиций — в константе $COUNT$. Поэтому коалиции из файла R_min_name должны быть подмножествами множества $\{0, 1, \ldots, N-1\}$, т. е. номера участников из файла

R_min_name должны находиться в диапазоне от 0 до $N-1$. Понятно, что можно работать не с константами N и $COUNT$, а с соответствующими переменными n и $count$, которые вычисляются на основе содержимого файла R_min_name. В этом случае функциям R_Rmin, Z_Zmax и т. д. нужно эти переменные передавать в качестве параметров. У читателя, интересующего этой темой, не должно возникнуть никаких трудностей с этим подходом. В нашей реализации мы оставляем константы, так как основная идея этой работы — показать пример реализации.

Если нужно восстановить секрет по файлам-долям произвольной правомочной коалиции, то можно закомментировать инструкцию

```
printf("Secret_sharing_rezult = %d\n",
        Secret_sharing_file(fsecret_name, dirname, C, m));
```

и запустить программу. В этом случае по файлам-долям, которые находятся в директории dirname, файл-секрет будет восстановлен. Заметим, что при успешном разделении файла-секрета на экране должно быть напечатано значение 0.

Если необходимо содержимое множеств коалиций (после их построения), например множеств R и Z, записать в файл, то можно в функцию main добавить строки

```
PrintFile("d:\\R.txt", R, nr);
PrintFile("d:\\Z.txt", Z, nz);
```

Например, если $N = 7$, $COUNT = 127$ и файл R_min_name содержит множество минимальных правомочных коалиций

$$\{0,1,5\},\quad \{2,3,5\},\quad \{2,4\},\quad \{4,6\},$$

записанных построчно и без скобок, то в файл "d:\R.txt"будут записаны все следующие правомочные коалиции (построчно и без скобок):
{2, 4}, {0, 2, 4}, {1, 2, 4}, {0, 1, 2 4}, {2, 3, 4}, {0, 2, 3, 4}, {1, 2, 3, 4}, {0, 1, 2, 3, 4}, {0, 1, 5}, {0, 1, 2, 5}, {0, 1, 3, 5}, {2, 3, 5}, {0, 2, 3, 5}, {1, 2, 3, 5}, {0, 1, 2, 3, 5}, {0, 1, 4, 5}, {2, 4, 5}, {0, 2, 4, 5}, {1, 2, 4, 5}, {0, 1, 2, 4, 5}, {0, 1, 3, 4, 5}, {2, 3, 4, 5}, {0, 2, 3, 4, 5}, {1, 2, 3, 4, 5}, {0, 1, 2, 3, 4, 5}, {4, 6}, {0, 4, 6}, {1, 4, 6}, {0, 1, 4, 6}, {2, 4, 6}, {0, 2, 4, 6}, {1, 2, 4, 6}, {0, 1, 2, 4, 6}, {3, 4, 6}, {0, 3, 4, 6}, {1, 3, 4, 6}, {0, 1, 3, 4, 6}, {2, 3, 4, 6}, {0, 2, 3, 4, 6}, {1, 2, 3, 4, 6}, {0, 1, 2, 3, 4, 6}, {0, 1, 5, 6}, {0, 1, 2, 5, 6}, {0, 1, 3, 5, 6}, {2, 3, 5, 6}, {0, 2, 3, 5, 6}, {1, 2, 3, 5, 6}, {0, 1, 2, 3, 5, 6}, {4, 5, 6}, {0, 4, 5, 6}, {1, 4, 5, 6}, {0, 1, 4, 5, 6}, {2, 4, 5, 6}, {0, 2, 4, 5, 6}, {1, 2, 4, 5, 6}, {0, 1, 2, 4, 5, 6}, {3, 4, 5, 6}, {0, 3, 4, 5, 6}, {1, 3, 4, 5, 6}, {0, 1, 3, 4, 5, 6}, {2, 3, 4, 5, 6}, {0, 2, 3, 4, 5, 6}, {1, 2, 3, 4, 5, 6}, {0, 1, 2, 3, 4, 5, 6}.

При это в файл "d:\Z.txt"будут записаны следующие все неправомочные коалиции (построчно и без скобок):
{0}, {1}, {0, 1}, {2}, {0, 2}, {1, 2}, {0, 1, 2}, {3}, {0, 3}, {1, 3}, {0, 1, 3}, {2, 3}, {0, 2, 3}, {1, 2, 3}, {0, 1, 2, 3}, {4}, {0, 4}, {1, 4}, {0, 1, 4}, {3, 4}, {0, 3, 4}, {1, 3, 4}, {0, 1, 3, 4}, {5}, {0, 5}, {1, 5}, {2, 5}, {0, 2, 5}, {1, 2, 5}, {3, 5}, {0, 3, 5}, {1, 3, 5}, {4, 5}, {0, 4, 5}, {1, 4, 5}, {3, 4, 5},

{0, 3, 4, 5}, {1, 3, 4, 5}, {6}, {0, 6}, {1, 6}, {0, 1, 6}, {2, 6}, {0, 2, 6}, {1, 2, 6}, {0, 1, 2, 6}, {3, 6}, {0, 3, 6}, {1, 3, 6}, {0, 1, 3, 6}, {2, 3, 6}, {0, 2, 3, 6}, {1, 2, 3, 6}, {0, 1, 2, 3, 6}, {5, 6}, {0, 5, 6}, {1, 5, 6}, {2, 5, 6}, {0, 2, 5, 6}, {1, 2, 5, 6}, {3, 5, 6}, {0, 3, 5, 6}, {1, 3, 5, 6}.

Итак, после разделения файла-секрета (с указанными выше параметрами N, $COUNT$, R_min) на файлы-доли будет создана директория "d:\fractions_of_a_secret", в которую будут записаны файлы fraction0, fraction1,..., fraction6. Оставим в директории dirname файлы-доли любой правомочной коалиции (из файла "d:\R.txt"), например файлы-доли fraction0, fraction1, fraction5. Тогда при восстановлении файла-секрета все пройдет успешно и он будет восстановлен. Но если собирается любая неправомочная коалиция, например, с долями fraction0, fraction1, fraction3, то файл-секрет будет восстановить невозможно.

Заметим, что вычислительная сложность построения множеств R, Z, R_min, Z_max составляет $O(2^n)$ операций, где n — число участников, так как перебираются все возможные коалиции участников, хоть и эффективным образом с использованием битовых операций. Но после построения данных множеств сам процесс разделения секрета и его восстановления умеет уже другую сложность вычислений. Для разделения секрета s на доли на основе кумулятивного массива C размером $n \times m$ понадобится $O(nm)$ операций. Пусть l — размерность файла-секрета. Тогда общая вычислительная сложность разделения файла-секрета составит $O(nml)$ операций. Пусть r — число участников некоторой правомочной коалиции, которая собирается восстановить секрет. В этом случае вычислительная сложность восстановления файла-секрета составит $O(rml)$ операций.

Также заметим, что простота восстановления секрета в данной схеме позволяет при небольшом объеме долей восстановить секрет без использования ПК, похожий пример приведен в предыдущей схеме.

5. Программная реализация совершенной и идеальной схемы на основе разбиения множества участников

Пусть $\{1, 2, \ldots, n\} = X_1 \cup \ldots \cup X_t$ — некоторое разбиение множества участников $\{1, 2, \ldots, n\}$. Построим совершенную и идеальную схему разделения секрета, для которой коалиция $X \subseteq \{1, \ldots, n\}$ является правомочной тогда и только тогда, когда $X \cap X_i \neq \varnothing$ для любого $i = 1, \ldots, t$, т. е. коалиция X содержит хотя бы одному участнику из каждого множества X_i, $i = 1, \ldots, t$. Обозначим $n_i = |X_i|$, $i = 1, \ldots, t$. Без ограничения общности будем считать, что

$$\begin{aligned} &X_1 = \{1,\ 2, \ldots,\ n_1\}, \\ &X_2 = \{n_1 + 1,\ n_1 + 2, \ldots,\ n_1 + n_2\}, \\ &\ldots \\ &X_t = \{n_1 + n_2 + \ldots + n_{t-1} + 1,\ n_1 + n_2 + \ldots + n_{t-1} + 2, \ldots,\ n\}. \end{aligned} \tag{1}$$

Пусть G — некоторая аддитивная абелева группа и требуется между n участниками разделить секрет $s \in G$. Сгенерируем случайным (равновероятным) образом $t - 1$ элементов

$\alpha_1, \ldots, \alpha_{t-1} \in G$, а элементу α_t придадим значение $\alpha_t = s - \alpha_1 - \ldots - \alpha_{t-1}$. После этого каждому участнику из множества X_i передадим долю α_i, $i = 1, \ldots, t$. Для восстановления секрета достаточно хотя бы по одному участнику из каждого из множеств $X_1, \ldots, X_t$ предоставить свои доли и вычислить $s = \alpha_1 + \ldots + \alpha_t$. Заметим, что если доли $\alpha_1, \ldots, \alpha_{t-1}$ вырабатываются равновероятным образом, то полученная схема будет совершенной. При этом схема является идеальной.

В качестве группы G в реализации будет выступать аддитивная группа поля $GF(2^8)$ (иными словами, множество двоичных длины 8 с поразрядной операцией сложения по модулю два).

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<string.h>
#include <dirent.h>
#include <sys/types.h>

#define FRACTION_NAME "fraction" // имена файлов долей секрета
#define FILE_LEN 512 // максимальная длина полного имени файла

#define BUFSIZE 4048 // размер буфера обмена

typedef unsigned char uint8;
typedef unsigned short uint16;
typedef unsigned long long uint64;

// Размер файла в байтах
uint64 Size(const char *fileName)
{
   FILE *f;
   uint64 size;
   if ((f = fopen(fileName, "rb")) == NULL)
      return 0;
   fseek(f, 0, SEEK_END); // перемещаем указатель в конец файла
   size = ftell(f);       // считываем текущую позицию указателя
   fclose(f);
   return size;
}

// Разделение секрета s на t долей,
// которые записываются в массив y
```

```
void Secret_sharing(uint8 s, uint8 *y, long t)
{
    long i;

    y[t - 1] = s;
    // Здесь для примера используется небезопасный генератор rand()
    for(i = 0; i < t - 1; i++)
    {
        y[i] = rand() % 256;
        y[t - 1] ^= y[i];
    }
    // В итоге, y[t-1] = s - (y[0]+...+y[t-2])
}

// Восстановление секрета s по t долям y[0],...,y[t-1]
uint8 Restoring_secret(uint8 *y, long t)
{
    long i;
    uint8 s = 0;
    // Восстановление секрета с использованием дискретных логарифмов
    for(i = 0; i < t; i++)
        s ^= y[i];
    return s;
}

// Выделение памяти для двумерного динамического массива-буфера
// размера n на BUFSIZE для файлов-долей секрета
uint8 **Allocate(long n)
{
    uint8 **a;
    long i;
    a = (uint8 **)malloc(n*BUFSIZE*sizeof(uint8) + n*sizeof(uint8 *));
    if (a != NULL)
        for(i = 0; i < n; i++)
             a[i] = (uint8 *)(a + n) + i*BUFSIZE;
    return a;
}

// Разделение файла-секрета fname на t файлов-долей.
// Файлы-доли секрета будут помещены в директорию dirname
int Secret_sharing_file(char *fname, char *dirname, long t)
```

```
{
    FILE *fsecret, // файл-секрет
        **f; // файлы-доли секрета
    char name[FILE_LEN], st[FILE_LEN];
    long i, j, len, count;
    uint8 s; // секрет
    uint8 *y; // доли секрета
    uint8 *bufs, **bufy; // буферы обмена для файла-секрета и файлов-долей

    // Выделяем место для буферов обмена
    bufs = (uint8 *)malloc(BUFSIZE*sizeof(*bufs));
    bufy = Allocate(t);
    if (bufs == NULL || bufy == NULL)
        return 3;

    // Создаем директорию для файлов-долей
    mkdir(dirname);
    // Открываем файл-секрет для чтения
    if ((fsecret = fopen(fname, "rb")) == NULL)
        return 10;
    // В переменной name будут по-очереди записаны полные имена файлов-долей секрета.
    // Изначально в строковую переменную  name записываем полное название
    // директории dirname и добавляем в конец "\fraction".
    strcpy(name, dirname);
    strcat(name, "\\");
    strcat(name, FRACTION_NAME);
    len = strlen(name);
    // Динамический массив файлов-указателей-долей секрета
    f = (FILE **)malloc(t * sizeof(*f));
    // Динамический массив для долей секрета
    y = (uint8 *)malloc(t * sizeof(*y));
    if (f == NULL || y == NULL)
        return 20;
    // Открываем файлы-доли секрета для записи
    for(i = 0; i < t; i++)
    {
        itoa(i + 1, st, 10); // переводим номер участника в строку st
        strcpy(name + len, st); // добавляем номер к названию файла-доли секрета
        f[i] = fopen(name, "wb"); // открываем файл-долю для записи
        if (f[i] == NULL)
            return 30;
```

```
    }

    // Разделяем файл-секрет на доли, которые записываем в файлы-доли секрета,
    // используя для ускорения работы буфер обмена bufs
    while (count = fread(bufs, sizeof(*bufs), BUFSIZE, fsecret))
    {
        for(j = 0; j < count; j++)
        {
            // Вычисляем доли y секрета bufs[j]
            Secret_sharing(bufs[j], y, t);
            // Записываем доли секрета bufs[j] в буфер bufy
            // для файлов-долей секрета
            for(i = 0; i < t; i++)
                bufy[i][j] = y[i];
        }
        // Накопленные в буфере bufy доли секрета записываем в файлы-доли
        for(i = 0; i < t; i++)
           fwrite(bufy[i], sizeof(uint8), count, f[i]);
    }

    fclose(fsecret);
    for(i = 0; i < t; i++)
        fclose(f[i]);
    free(y); free(f); free(bufs); free(bufy);
    return 0;
}

// Восстановление файла-секрета по t файлам-долей
int Restoring_secret_file(char *fname, char *dirname, int t)
{
    FILE *fsecret, // файл-секрет
        **f; // файлы-доли секрета
    struct dirent *entry; // для "вытаскивания" файлов-долей секрета из директории
    DIR *dir; // указатель-директория
    char name[FILE_LEN], st[FILE_LEN];
    long i, len;
    uint8 *y;
    uint64 j, k, size, q, r;
    uint8 *bufs, **bufy; // буферы обмена для файла-секрета и файлов-долей

    // Выделяем место для буферов обмена
```

```
bufs = (uint8 *)malloc(BUFSIZE*sizeof(*bufs));
bufy = Allocate(t);
if (bufs == NULL || bufy == NULL)
    return 3;

// Открываем директорию, откуда будем брать файлы-доли секрета
if ((dir = opendir(dirname)) == NULL)
    return 5;
// В переменной name будут по-очереди записаны полные имена файлов-долей секрета
strcpy(name, dirname);
strcat(name, "\\");
len = strlen(name);
// Создаем динамические массивы:
// y - доли секрета,
// f - файловые указатели-доли секрета
y = (uint8 *)malloc(t * sizeof(*y));
f = (FILE **)malloc(t * sizeof(*f));
if (y == NULL || f == NULL)
    return -10;
i = 0;
// Считываем все файлы из директории dirname
while ((entry = readdir(dir)) != NULL && i < t)
{
    // Ищем в директории те файлы, которые содержат название "fraction"
    if (!strncmp(FRACTION_NAME, entry->d_name, strlen(FRACTION_NAME)))
    {
        strcpy(name + len, entry->d_name);
        // Открываем файлы-доли для чтения
        if ((f[i] = fopen(name, "rb")) == NULL)
            return 20;
        // Проверяем, что все файлы-доли имеют одинаковый размер
        if (i == 0)
            size = Size(name);
        else if(size != Size(name))
            return 30;
        i++;
    }
}
// если файлов-долей секрета < t, то секрет нельзя восстановить
if (i < t)
    return 40;
```

```
// Открываем файл-секрет для записи
if ((fsecret = fopen(fname, "wb")) == NULL)
    return 10;

// Вычисляем, сколько целых буферов обмена помещается в файл
q = size / BUFSIZE;
r = size - q * BUFSIZE;

// Считываем доли из файлов-долей и восстанавливаем файл-секрет
for(j = 0; j < q; j++)
{
    for(i = 0; i < t; i++)
        fread(bufy[i], sizeof(uint8), BUFSIZE, f[i]);
    for(k = 0; k < BUFSIZE; k++)
    {
        for(i = 0; i < t; i++)
            y[i] = bufy[i][k];
        bufs[k] = Restoring_secret(y, t);
    }
    fwrite(bufs, sizeof(*bufs), BUFSIZE, fsecret);
}
if (r > 0)
{
    for(i = 0; i < t; i++)
        fread(bufy[i], sizeof(uint8), r, f[i]);
    for(k = 0; k < r; k++)
    {
        for(i = 0; i < t; i++)
            y[i] = bufy[i][k];
        bufs[k] = Restoring_secret(y, t);
    }
    fwrite(bufs, sizeof(*bufs), r, fsecret);
}

closedir(dir);
fclose(fsecret);
for(i = 0; i < t; i++)
    fclose(f[i]);
free(y); free(f); free(bufs); free(bufy);
return 0;
```

```
}

int main()
{
    long t;
    long i;
    char dirname[] = "d:\\fractions_of_a_secret"; // директория для файлов-долей
    char fsecret_name[] = "d:\\a.pdf"; // файл-секрет
    char fsecret_name2[] = "d:\\b.pdf"; // файл-секрет после восстановления

    srand(time(NULL));

    t = 7;  // задаем значение t
    // Разделяем файл-секрет на файлы-доли
    printf("Secret_sharing_rezult = %d\n",
            Secret_sharing_file(fsecret_name, dirname, t));
    // Восстанавливаем файл-секрет на основе файлов-долей
    printf("Restoring_secret_rezult = %d\n",
            Restoring_secret_file(fsecret_name2, dirname, t));

    return 0;
}
```

Перед запуском программы нужно сначала зафиксировать некоторое разбиение множества участников $\{1, 2, \ldots, n\}$ вида (1). После того, как значение t определено, это значение нужно записать в переменную t.

При первом запуске программы можно закомментировать инструкцию

```
printf("Restoring_secret_rezult = %d\n",
        Restoring_secret_file(fsecret_name2, dirname, t));
```

В этом случае будет произведено разделение файла-секрета на файлы-доли, которые будут записаны в директорию dirname. Например, после запуска программы с указанным значением $t = 7$ будет создана директория "d:\fractions_of_a_secret", в которую будут записаны файлы fraction1, fraction2,..., fraction7. Заметим, что при успешном разделении файла-секрета на экране должно быть напечатано значение 0. Теперь файл fraction1 передается всем участникам множества X_1. Файл fraction2 передается всем участникам множества X_2. Аналогично происходит и с остальными файлами-долями fraction3,..., fraction7.

Если нужно восстановить секрет по t файлам-долям, то можно закомментировать инструкцию

```
printf("Secret_sharing_rezult = %d\n",
        Secret_sharing_file(fsecret_name, dirname, n, t));
```

В этом случае по файлам-долям, которые находятся в директории dirname, файл-секрет будет восстановлен. Для этого нужно, чтобы все t файлов-долей находились в директории dirname. Это означает, что у собравшихся вместе участников имеются все эти доли.

Заметим, что простота восстановления секрета в данной схеме позволяет при небольшом объеме долей восстановить секрет без использования ПК. Приведем пример. Пусть $n = 10$ — число участников, а разбиение множества участников имеет вид

$$\{1, 2, \ldots, 10\} = \{1, 2\} \cup \{3, 4, 5\} \cup \{6, 7, 8\} \cup \{9, 10\}.$$

Поэтому $t = 4$ — число частей разбиения (такое значение определяется при запуске программы). Обозначим

$$X_1 = \{1, 2\}, \quad X_2 = \{3, 4, 5\}, \quad X_3 = \{6, 7, 8\}, \quad X_4 = \{9, 10\}.$$

Предположим, что каждый участник из множества X_i, $i = 1, 2, 3, 4$, получил следующую долю секрета:

X_1: 11000110 01111100 11111100 11111010 10101110,
X_2: 10000101 00011111 00010111 00110001 10111011,
X_3: 11000110 01010110 11001111 01011000 00010001,
X_4: 10000111 00110110 00100001 10010100 00001111.

Заметим, что эти значения можно получить с помощью функции Print_file из параграфа 3, где более подробно об этом говорится. Каждая из представленных долей содержит пять двоичных векторов длины 8. Это значит, что секрет состоит из 5 байт. Предположим, что собрались вместе хотя бы по одному представителю из каждого множества X_i, например, участники $\{1, 4, 8, 9\}$. У них в совокупности имеются все 4 доли секрета.

После поразрядного сложения векторов (первые 8-битные векторы у каждого участника) 11000110, 10000101, 11000110, 10000111 по модулю два участники получают значение 00000010=2.

После поразрядного сложения векторов (вторые 8-битные векторы у каждого участника) 01111100, 00011111, 01010110, 00110110 по модулю два участники получают значение 00000011=3.

И так далее. В итоге, участники получают значение секрета 2, 3, 5, 7, 11.

Этот пример наглядно иллюстрирует, что для небольшого объема долей и значения t восстановить секрет можно без использования дополнительных вычислительных средств. Заметим, что для более компактного хранения долей секрета (на бумаге) можно хранить значения, например, в шестнадцатеричном виде. В этом случае у групп участников будут следующие доли:

X_1: c6 7c fc fa ae,
X_2: 85 1f 17 31 bb,
X_3: c6 56 cf 58 11,
X_4: 87 36 21 94 0f.

Заключение

В работе приведена программная реализация схемы разделения секрета Шимира на основе конечного поля $GF(2^8)$, схемы на основе равновесных двоичных кодов и схемы Ито—Саито—Нишизеки на основе аддитивной группы поля $GF(2^8)$. Значимость этих схем состоит в том, что если использовать случайный генератор с равномерным распределением на множестве $GF(2^8)$ и с помощью него генерировать коэффициенты многочлена Лагранжа в схеме Шамира или доли вспомогательной (m, m)-схемы в схеме на основе равновесных кодов или в схеме Ито—Саито—Нишизеки, то данные три схемы будут являться совершенными, т. е. не зависеть от вычислительных возможностей злоумышленника.

Список литературы

1. Рацеев С. М. *Математические методы защиты информации* : учебное пособие для вузов. СПб.: Лань, 2022. 544 с.

2. Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process. National Institute of Standards and Technology Interagency or Internal Report. NIST IR 8413-upd1. July 2022, 102 pages. https://csrc.nist.gov/publications/detail/nistir/8413/final.

3. McEliece R. J., Sarwate D. V., On sharing secrets and Reed—Solomon codes // *Comm. ACM.* 1981, v. 24, p. 583–584.

4. Massey J. L. Minimal codewords and secret sharing // *In Proceedings of the 6-th Joint Swedish-Russian Workshop on Information Theory*, Molle, Sweden, August 1993. P. 269–279.

5. Massey J. L. Some applications of coding theory in cryptography // *Codes and Ciphers: Cryptography and Coding IV*. 1995, p. 33–47.

6. Рацеев С. М. *Элементы высшей алгебры и теории кодирования*: учебное пособие для вузов. СПб.: Лань, 2022. 656 с.

7. Рацеев С. М., Убанеева Е. Г. О реализации конечных полей характеристики два и об их применении в криптосистемах // *Ученые записки УлГУ. Сер. Математика и информационные технологии.* 2022, № 2, с. 90–107.

8. Рацеев С. М. Эффективная реализация структур доступа для схем разделения секрета // *Ученые записки УлГУ. Сер. Математика и информационные технологии.* 2022, № 2, с. 75–89.

9. Рацеев С. М. *Программирование на языке Си*: учебное пособие для вузов. СПб.: Лань, 2022. 332 с.

On implementation of some perfect secret sharing schemes

Ratseev, S. M.

ratseevsm@mail.ru
Ulyanovsk State University, Russia

In the paper a programming implementation of Shamir secret sharing scheme over the finite field of characteristic two, schemes based on equilibrium binary codes and Ito—Saito—Nishizeki schemes are investigated. The first two schemes are threshold secret sharing schemes, the third is a scheme for a general access structure. The paper is educational and methodological in nature and can help with the programming implementation of secret sharing schemes.

Keywords: *secret sharing schemes, access structure, Shamir secret sharing scheme, Ito—Saito—Nishizeki secret sharing scheme*