Министерство образования Республики Беларусь

Учреждение образования

«Полоцкий государственный университет»

Д.Ф. Пастухов, Ю.Ф. Пастухов, И.С. Глебко

ПОЛИНОМИАЛЬНОЕ КОДИРОВАНИЕ.

Учебное пособие к лекционным и практическим занятиям

для студентов специальности

1-98 01 01 Компьютерная безопасность

Новополоцк
ПГУ
2019

УДК 519.72

Рецензенты:
А.А. Козлов, кандидат физико-математических наук, доцент, заведующий кафедрой Высшей математики и дифференциальных уравнений Полоцкого государственного университета;

Пастухов Д.Ф., Пастухов Ю.Ф., И.С. Глебко
Полиномиальное кодирование /Д.Ф. Пастухов, Ю.Ф. Пастухов, И.С.Глебко.- Новополоцк: ПГУ,2019. - 23 с.

В учебном пособии описан метод полиномиального кодирования. Для студентов программистов 98 01 01 (компьютерная безопасность).

Для студентов университетов, педагогических вузов, технических вузов, преподавателей, инженеров, программистов, использующих в своей практической деятельности математические методы шифрования.

Одобрено и рекомендовано к изданию
методической комиссией факультета информационных технологий
В качестве учебного пособия

Кафедра технологий программирования

Предисловие

Предлагаемое учебное пособие создано на основе работы авторов со студентами - дипломниками 4 курса специальности Компьютерная безопасность 98 01 01 (математические методы и программные системы) в Полоцком государственном университете. Были использованы идеи Лидовского В.В.(Теория информации). Полиномиальное кодирование использует многочлены и алгебру многочленов с коэффициентами из группы остатков целых чисел Z_2

Генератор - многочлен, коэффициенты которого являются секретными ключами, умножается на многочлен - слово, коэффициенты которого также остатки чисел по модулю 2. Вектор коэффициентов многочлена - слова из двоичной системы счисления можно переписать в десятичной системе счисления. Если отождествить полученное число с порядковым номером символа от 0 до 255 с символом алфавита ASCII. То для взаимно однозначной записи одного символа алфавита или текстовой информации достаточно одного байта двоичного кода (восемь нулей или (и) единиц, записанных соответствующим образом).

Шифрование полиномиальным кодом заключается в перемножении коэффициентов многочлена для символа сообщения на многочлен - генератор. Декодирование происходит симметрично в обратном порядке. Многочлен шифр делится на многочлен – генератор, получаем многочлен символ из 8 коэффициентов, который снова переводится в соответствующий символ.

Для ускорения шифрования и дешифрования, а также для криптостойкости шифра можно шифровать информацию пакетами по m=8*k бит одновременно, что соответствует шифрованию k символов одновременно. То есть 8*k бит нулей и единиц являются коэффициентами многочлена-сообщения.

Для контроля безошибочного шифрования нами использована хеш – функция. А именно, при делении каждого шифра на многочлен генератор, получается многочлен остаток степени не выше степени многочлена генератора. Если остаточный многочлен имеет нулевые коэффициенты, то декодирование произошло безошибочно. Иначе, существует простой алгоритм, позволяющий обнаружить и исправить ошибку кратности не выше двух. Расстояние между шифр-словами определяется по Хэммингу, оно равно числу соответствующих несовпадающих символов в словах одинакового размера. В качестве слова - генератора мы использовали квазисовершенный многочлен-генератор Голея и многочлены близкие к данному многочлену в том смысле, что минимальное расстояние между словами равно 5 либо 6(у многочлена Голея 7). Нами написана вспомогательная программа для отыскания минимального расстояния между словами в шифрованном пространстве слов, которое может иметь произвольную размерность. Поэтому мы шифровали строку английского алфавита малых и больших букв, а также цифры (порядка 60 символов) для соответствующего многочлена генератора в то же шифрованное пространство, что и передаваемое сообщение, а затем программой определялось минимальное и максимальное расстояния между шифр-словами вспомогательной алфавитной строки.

Отметим, что полиномиальное кодирование можно применять для хранения долговременных паролей-ключей с числом символов более 256, так как шифрование до 1150 символов(1 максимально допустимая строка символьной фразы в компиляторе C++) шифруется и дешифруется взаимно однозначно. Для увеличения массива фразы шифрования необходимо объединить несколько разных фраз в общий символьный массив.

Представлен код программы на языке С++. Для запуска программы нужно скопировать ее листинг из pdf файла, вставить в новый проект, перед запуском удалить номера страниц pdf файла.

Полиномиальное шифрование. Многочлен Голея. Оптимальные свойства многочлена Голея

Многочлен Голея имеет вид

$$y(x) = 1 + x + x^5 + x^6 + x^7 + x^9 + x^{11} \quad (1)$$

Символу h с номером 104 в ASCII соответствует двоичный код $([104])_{10} =$

$=(1101000)_2 =$ $=64+32+8=104$. Тогда шифр, полученный с помощью многочлена Голея - 00011101111001101110. Проверим, что указанный шифр, соответствующий слову – многочлену $x^6 + x^5 + x^3$ правильный.

При умножении многочленов символа и многочлена – генератора получим = $= (00011101111001101110)_2$

Из таблицы 1 видно, что минимальное попарное расстояние по Хэммингу между шифрами слов для 26 малых и больших букв английского алфавита (и цифр), поолученных совершенным многочленом Голея, равно семи. Такой код способен исправить ошибки кратности не выше 2k+1=7, то есть k=3 – тройные ошибки[1].

Таблица 1 – Попарные расстояния по Хэммингу для 26 малых и больших букв английского алфавита (и цифр), зашифрованных совершенным кодом многочлена Голея.

i/j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
0																										
1	8																									
2	8	12																								
3	8	8	8																							
4	8	8	8	8																						
5	7	7	7	7	11																					
6	7	7	7	7	7	8																				
7	8	8	8	8	8	7	7																			
8	8	8	8	8	8	7	11	8																		
9	7	7	11	7	7	12	8	11	7																	
10	7	7	7	7	11	8	8	7	7	8																
11	7	7	7	7	7	11	8	8	7	8	12															

12	11	7	7	7	7	8	8	11	7	8	8	8													
13	8	8	8	8	12	7	7	8	12	7	7	11	7												
14	7	7	7	11	7	8	12	7	7	8	8	8	8	7											
15	11	7	11	11	7	8	8	7	7	8	12	8	8	7	8										
16	8	8	8	8	8	7	7	8	8	11	7	11	7	8	11	7									
17	11	7	7	11	7	12	8	7	7	8	8	8	8	11	8	8	7								
18	8	8	12	8	8	11	11	8	8	7	11	7	7	8	7	7	8	7							
19	7	11	7	7	7	8	8	7	11	8	8	12	8	7	8	12	7	8	7						
20	12	8	8	8	8	7	11	8	8	7	11	7	7	8	7	7	12	7	8	7					
21	12	8	8	8	8	11	7	8	8	7	7	7	7	8	7	7	12	7	8	11	8				
22	7	7	7	7	7	8	8	7	11	8	8	8	12	7	8	8	7	8	11	8	7	11			
23	7	11	7	7	7	8	8	7	11	12	8	8	8	7	8	8	7	12	7	8	11	7	8		
24	7	11	7	7	11	8	8	7	7	8	8	8	8	7	12	8	7	8	7	8	7	11	8	8	
25	7	11	7	7	11	8	8	7	7	8	8	8	12	7	8	8	7	8	7	8	11	7	8	8	8

Например, при шифровании двенадцатый символ алфавита имеет шифр 0011010011110011110(s), а двадцать первый символ шифр 10100001110111111110(x). Сравнивая соответствующие позиции шифров, видно, что они отличаются в семи позициях, то есть расстояние между ними равно семь. Тот же результат показывает и таблица 1 dist(21,12)=7.

Пусть шифрованный двенадцатый символ получил три ошибки 1111110011110011110, тогда по Лидовскому[1] ближайший к ошибочному символу правильный символ 0011010011110011110 и по алгоритму кратчайшего расстояния, именно на правильный символ вернет программа ошибочный символ. То есть совершенный код обладает уникальным алгоритмом проверки и устранения ошибок тройной кратности для шифрованного текста.

Код Голея (формула 1) может обнаруживать ошибку k=d-1=7-1=6, т.е. кратности 6 и исправлять ошибку кратности k=(d-1)/2=(7-1)/2=3 [1].

Для этого достаточно в базе кодов шифрованных символов для букв английского алфавита, найти слово, удаленное на минимальное расстояние от ошибочного (не более 3-х символов) и заменить ошибочное слово на слово в базе данных. Процесс проверки исправления ошибок должен предшествовать процессу дешифрования.

Описание алгоритма ядра программы на языке С++

1) Алгоритм шифрования

В программе шифрования используется функция сложения двух чисел по модулю два (листинг 1).

Листинг 1. Программа сложения двух чисел по модулю два.

```
int sr(int a, int b)
{
  return (a+b)%2;
}
```

Коэффициенты многочлена двоичного кода {0,1} записываются в целочисленный массив mn[n+m+1], где n- степень кодирующего многочлена у которого число коэффициентов n+1, m-степень многочлена исходного слова с числом коэффициентов m+1. Отметим, что размерность массива превышает размерность пространства исходных слов, т.к. произведение многочлена сводится к матричному произведению, т.е. размерность кодирующего многочлена имеет максимальную из возможных (старшие коэффициенты равны нулю), а именно размерность пространства шифрованных слов. Степень произведения двух многочленов есть их сумма m+n, а число коэффициентов шифрованного слова равно m+n+1 (листинг 2).

Листинг 2. Обнуление верхних коэффициентов кодирования.

```
for(i=n+1;i<=n+m;i++)
 {
   mn[i]=0;
 }
```

Поскольку в данной работе используется методы шифрования символьного текста (русского, английского и т.д.), то для этого достаточно выбрать 256 символов клавиатуры. $256=2^8$, т.е. каждый символ можно переводить в двоичный код из нулей и единиц размером в один байт (8 нулей и единиц).

Исходная фраза записывается в символьную массив str[mm+1], где mm – количество символов в фразе. Эта же строка переводится в целочисленный массив (в консольном виде на рисунке 1).

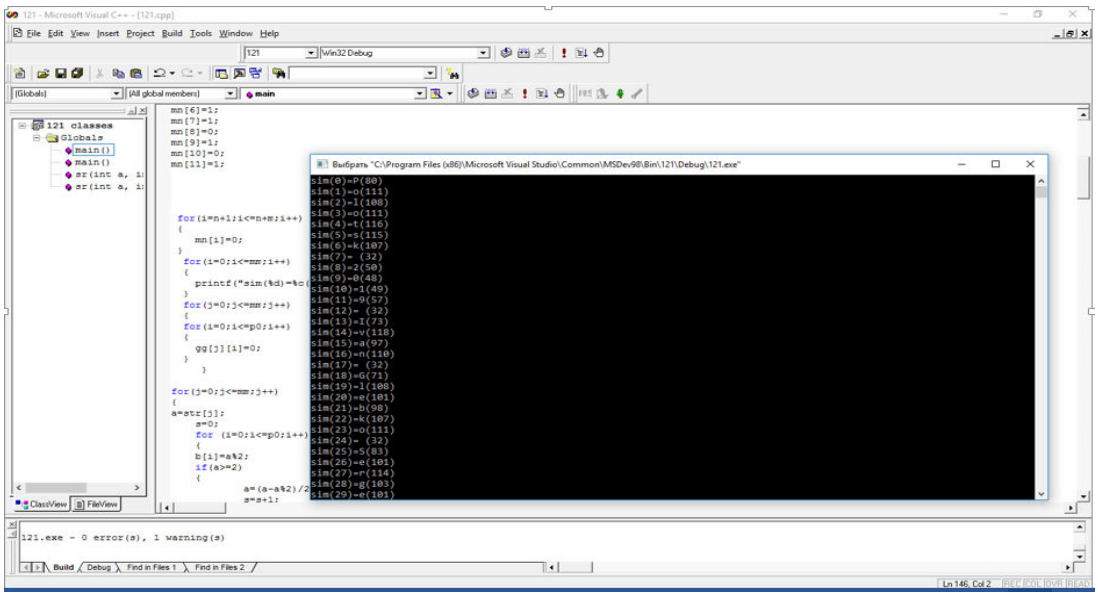

Рис. 1. Исходная фраза, предназначенная для шифрования.

Создаем двумерный массив двоичного кода исходной фразы, первоначально его обнуляем (листинг 3).

Листинг 3. Двумерный массив двоичного кода исходной фразы.

```
for(j=0;j<=mm;j++)
 { for(i=0;i<=p0;i++)
 {

   gg[j][i]=0;
 }
   }
```

Приводим по таблице ASCII символ сообщения в порядковый номер этого символа a=str[j]. Используя алгоритм Евклида в двоичной системе счисления переводим целое число «a» в вектор коэффициентов двоичного кода из нулей и единиц с помощью следующей программы (листинг 4) (Рис. 2).

Листинг 4. Алгоритм перевода символа в двоичный код.

```
for(j=0;j<=mm;j++)
{
a=str[j];
    s=0;
    for (i=0;i<=p0;i++)
    {
    b[i]=a%2;
    if(a>=2)
    {
                a=(a-a%2)/2;
                s=s+1;
                gg[j][i]=b[i];
                }
    if(a<=1)
    {
    b[i+1]=a;
    gg[j][i+1]=b[i+1];
    i=p0;
      }}}
```

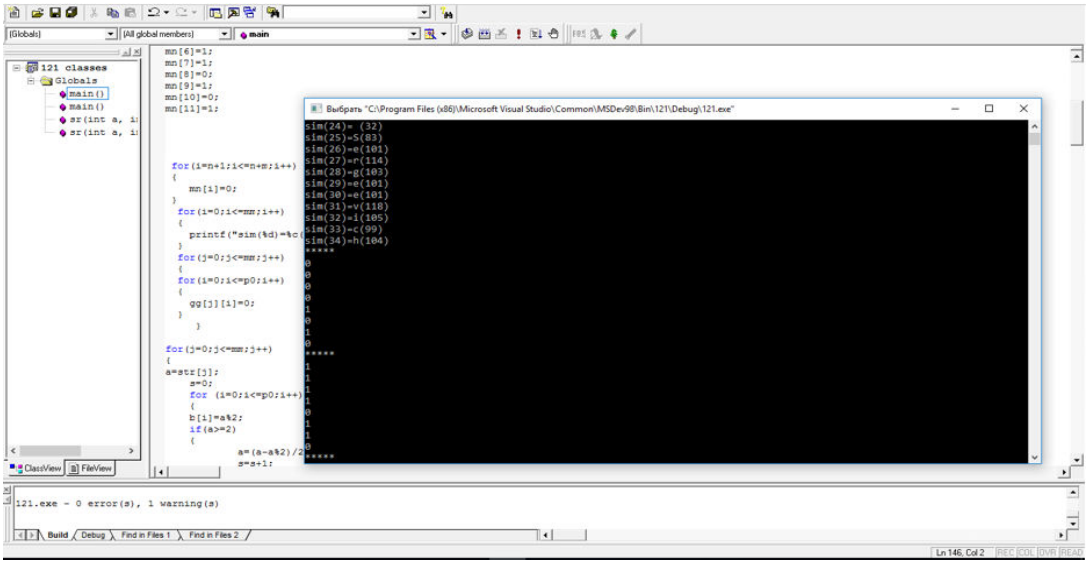

Рис. 2. Подпрограмма перевода номера символа в двоичный код согласно ASCII.

В матрице gg[j][i+1]=b[i+1], где j пробегает все символы сообщения от нуля до mm, а i проходит значения от 0 до p0=7. Матрица сообщения gg[j][i+1] содержит исходное сообщение в двоичном коде.

Далее полученный двоичный код сообщения можно вывести на экран (листинг 5).

Листинг 5. Вывод двоичного кода на экран.

```
for(j=0;j<=mm;j++)
{
printf("*****\n");
for(i=0;i<=p0;i++)
{
    printf("%d\n",gg[j][i]);
    }
    }
```

В векторный массив slovo[m+1] из матрицы сообщения по столбцам переводим очередной столбец матрицы сообщения.

Генерируем кодирующую матрицу g1[m+1][n+m+1] соответствующую многочлену-генератору. Первоначально матрица g1[i][j] обнуляется, далее многочлен-генератор из первой строки сдвигается вправо и вниз вдоль главной диагонали матрицы своим младшим коэффициентом (листинг 6).

Листинг 6. Образование вспомогательной матрицы умножения g1[i][j].

```
for(j=0;j<=n+m;j++)
    {
        g1[i][j]=0;
    }
    printf("%d %d\n",m,n+m);
    for(i=0;i<=m;i++)
    {
    for(j=i;j<=n+m;j++)
    {
g1[i][j]=mn[j-i];
    }
}
```

Контроль элементов кодирующей матрицы (листинг 7):

Листинг 7. Контроль элементов кодирующей матрицы.

```
for(i=0;i<=m;i++)
   for(j=0;j<=n+m;j++)
   {
      printf("g(%d,%d)=%d\n",i,j,g1[i][j]);
   }
```

Умножаем вектор коэффициентов двоичного кода из массива slovo[m+1] слева на кодирующую матрицу справа g1 порядка (m+1)*(m+n+1), образуя вектор шифра shifr[m+n] (листинг 8) (Рис. 3) .

Листинг 8. Образование вектора шифра shifr[m+n].

```
for(j=0;j<=n+m;j++)
   {
   sum=0;
   for(i=0;i<=m;i++)
   {
      sum=sum+slovo[i]*g1[i][j];
      sum=sum%2;
   }
      shifr[j] =sum;
      mas[d][j]=shifr[j]; }
```

Рис. 3. Программа вывода шифра многочленом Голея в пространстве шифра m+n.

Целочисленный размер шифра хранится в матрице mas[mm+1][n+m+1]. Текущий вектор шифра хранится в массиве shifr[j]. Для контроля массива шифра используем вывод (листинг 9).

Листинг 9. Вывод шифра на экран.

```
for(i=0;i<=mm;i++)
   {
   for(j=0;j<=nn;j++)
   {
      printf(" shifr(%d %d)=%d\n",i,j,mas[i][j]);
   }
   printf("\n");
   }
```

```
    printf("****\n");

for(p=0;p<=mm;p++)
{
    printf("p=%d\n",p);

for(j=0;j<=nn;j++)
{
    ee[j]=mas[p][j];
}
```

Шифрование завершается проверкой массива. Из приведенного описания видно, что в программе ведется контроль всех операций, которые можно использовать как протокол шифрования.

2) Описание алгоритма дешифрования

Алгоритм дешифрования составлен в общем виде для алгоритма Евклида, делением многочлена на многочлен. Исходный вектор шифра содержится в массиве ee[nn+1], nn=n+m (Рис. 4).

В алгоритме Евклида многочлен-генератор хранится в массиве mn[m+n+1]. Многочлен-остаток хранится в массиве ost[nn+1].

В начале программы коэффициенты многочлена-остатка обнуляются, аналогично обнуляются коэффициенты многочлена-частного. Массив chas[nn-n], степень многочлена частного не превышает nn-n=m.

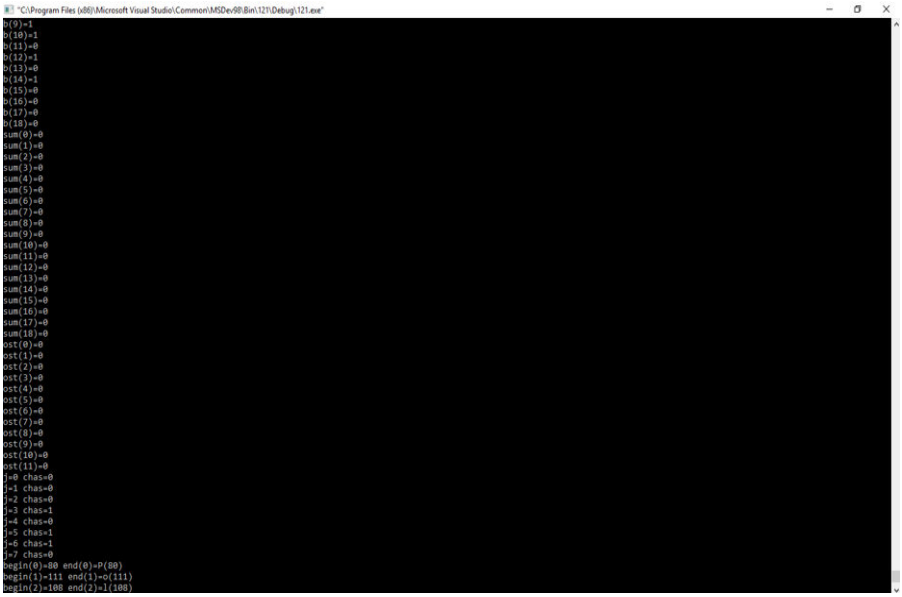

Рис. 4. Вывод двоичного кода символа при дешифровании в массиве chas[nn-n]=chas[m]

Вектор baza[nn+1] хранит коэффициенты вектора-сдвига, он также обнуляется.

Необходимо посчитать текущую размерность многочлена-остатка. Первоначальную роль многочлена-остатка выполняет многочлен-шифр (листинг10).

Листинг 10. Вычисление степени многочлена-остатка.

```
s=0;
for(i=0;i<=nn;i++)
```

```
{
   if(ee[nn-i]==0)

   {
   s=s+1;
       }
       else if(ee[nn-i]==1)
       {
       l=nn-s;
       i=nn;
       }
}
k=l-n;
```

Другими словами, определяется старший ненулевой коэффициент текущего делимого многочлена. Тогда степень многочлена - остатка равна k=l-n, где l - это текущая степень многочлена-делимого. Как только k<=0(степень текущего многочлена делимого<=0), программа выходит из цикла, а степень текущего остатка равна <=n.

Перевод двоичного кода дешифрованного символа в соответствующий символ клавиатуры (листинг 7) (Рис. 5).

Листинг 7. Преобразование двоичного кода многочлена – частного в символ клавиатуры.

```
for (j = 0; j <= mm; j++)
   {
      z = 0;
      h = 1;
      for (i = 0; i <= p0; i++)
      {
            z = z + gg[j][i] * h;
            h = h * 2;
      }
      end[j] = z;
   }
   for (j = 0; j <= mm; j++)
   {
      printf("begin(%d)=%d end(%d)=%c(%d)\n", j, str[j], j, end[j], end[j]);
   }
}
```

Рис. 5. Преобразование двоичного кода многочлена - частного в символ клавиатуры, сравнение качества дешифрования и проверка хеш-функции.

На Рис. 5 показан двоичный код дешифрованного последнего символа исходной фразы h (00010110) = $1*2^3+2^5+2^6$= 9+32+64=104. Так же на Рис.5 видно, что последний тридцать четвертый символ в фразе имеет порядковый номер в таблице ASCII 104 и соответствует букве h. Кроме того все символы исходной фразы включая пробел и знаки табуляции – посимвольно совпадают с дешифрованной фразой, т.е. совпадают порядковые номера символов исходной и конечной фраз по таблице ASCII.

Особенностью данной программы дешифрования является построение проверочной хеш-функции, значение которой в двоичном коде хранится в массиве ost[nn]. Из Рис. 7 видно, что последний символ и все остальные символы имеют нулевой вектор остатка, что соответствует делению нацело многочлена шифра на многочлен-генератор (его двенадцать коэффициентов являются ключами). Если вектор остатка содержит одну и более единиц, то дешифрование (деление нацело многочленов) проведены для соответствующего символа неверно.

Вспомогательная программа

На Рис. 6 изображен результат выполнения программой поиска попарного расстояния между шифрованными словами многочленом Голея, написанной в среде visual studio C++. Из нижней треугольной таблицы видно, что минимальное расстояние между словами не превышает семи, что подтверждает основные положения теории [1].

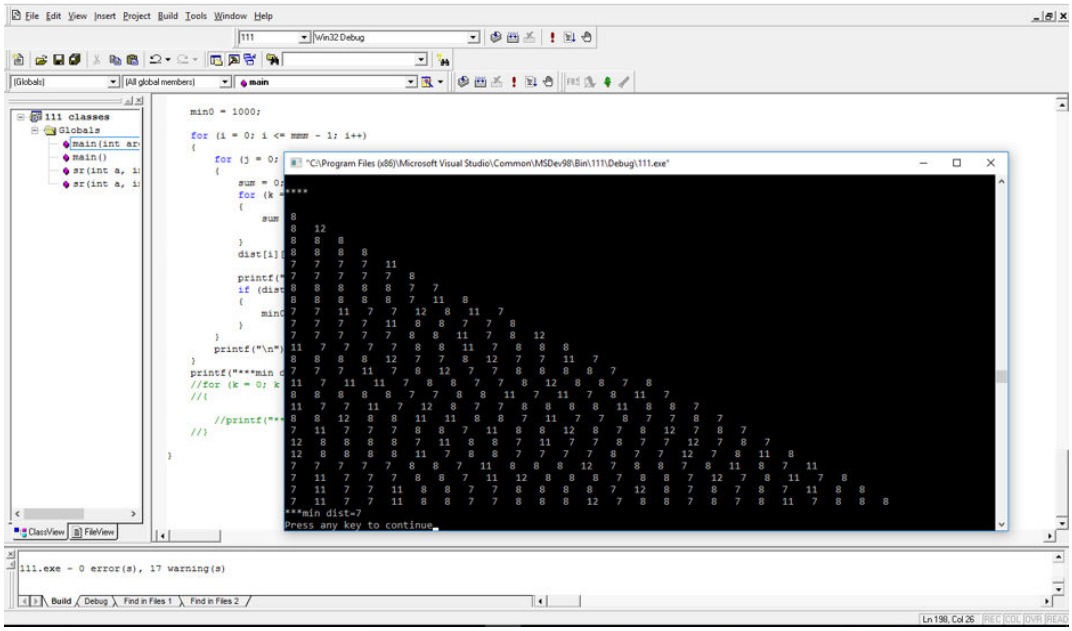

Рис. 6. Вспомогательная программа определения попарного расстояния между шифрованными словами в виде нижней треугольной матрицы.

Оценка криптостойкости шифра

Оценим криптостойкость описанного шифра. Пространство шифрованных слов имеет размерность равную 19 для многочлена 11 степени. Т.е. число различных многочленов степени не выше 18 равно 2^{19}=524288, т.е. порядка одного миллиона. Число различных многочленов степени 11 с различными коэффициентами равно 2^{12}=4096 многочленов. Тогда для дешифрования одного символа перебором нужно провести не более чем 524288* 4096=2147483648 попыток (испытаний). Для шифра пакета из пяти символов имеем число переборов2^{52}*4096=1.84*10^{19}

Если короткая фраза состоит из 6 различных символов, то необходимо число вариантов 256^6 =2,8*10^{14} . Что значительно меньше числа 1.84*10^{19}. Поэтому слово по его шифру определить быстрее перебором символами клавиатуры, чем перебрать все ключи из пространства ключей.

Таблица многочленов-генераторов

Запишем 15 разных многочленов, близких по минимальному расстоянию к многочлену Голея, экспериментально найденных вспомогательной программой.
Таблица 2. Таблица многочленов с минимальным расстоянием между шифрами >=5.

Коэффициенты многочлена	Минимальное расстояние	Максимальное расстояние между словами	Кратность исправляемой ошибки	Кратность обнаруживаемой ошибки
110001110101(Голея)	7	15	3	6
100001110101	6	14	2	5
101001110101	5	14	2	4
011001110101	5	13	2	4
110101110101	6	12	2	5
100011110101	5	14	2	4
110001110101	6	14	2	5
110111110101	5	14	2	4
011011110101	6	14	2	5
010011110101	5	14	2	4
110011110101	6	14	2	5
100000110101	5	13	2	4
010000110101	5	14	2	4
100100110101	6	14	2	5
101100110101	5	15	2	4

Кратность обнаружения ошибки $k_1 = d - 1$, кратность исправления ошибки $k_2 = \frac{d-1}{2}$ [1].

Пример работы программы совместно графическим интерфейсом

На рисунке 7 показан пример шифрования фразы «Polotsk 2019 Ivan Glebko Sergeevich», которую предварительно с помощью клавиатуры вводят в левое нижнее окно. Символы фразы могут быть любыми из 256 символов клавиатуры. Сравним Рис.7 и Рис.4. Первая строка шифра имеет код 0000111101101000010, этот шифр соответствует кодированию многочленом Голея большой буквы «P» английского шрифта. Сравним так же шифры конечных символов фразы Рис.7 и Рис.4.

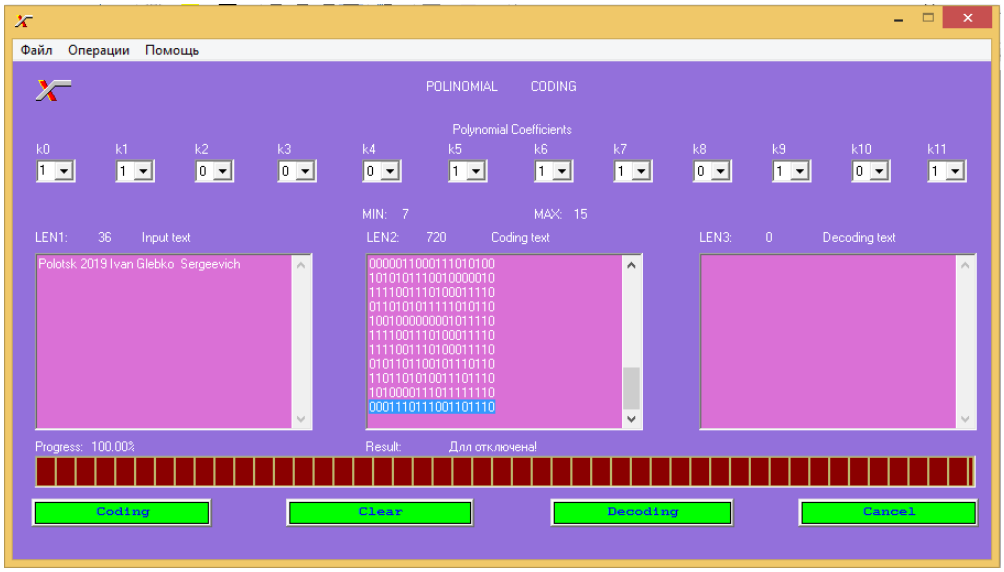

Рис. 7. Пример шифрования фразы «Polotsk 2019 Ivan Glebko Sergeevich».

На Рис.7 последнему символу h с номером ASCII 104 соответствует шифр 0001110111001101110. Проверим, что полученный шифр является истинным. Двоичный

код числа $([104)]_{10} = (1101000)_2 =$ =64+32+8=104.

При умножении = $(000111011100)_2$

Из Рис. 7 видно последняя строка (последний символ) шифрованной фразы совпадает с приведенными вычислениями.

Как видно из Рис.8, текст который изначально шифруется, после кодирования и декодирования – остается посимвольно неизменным, при этом в данном примере используются знаки табуляции.

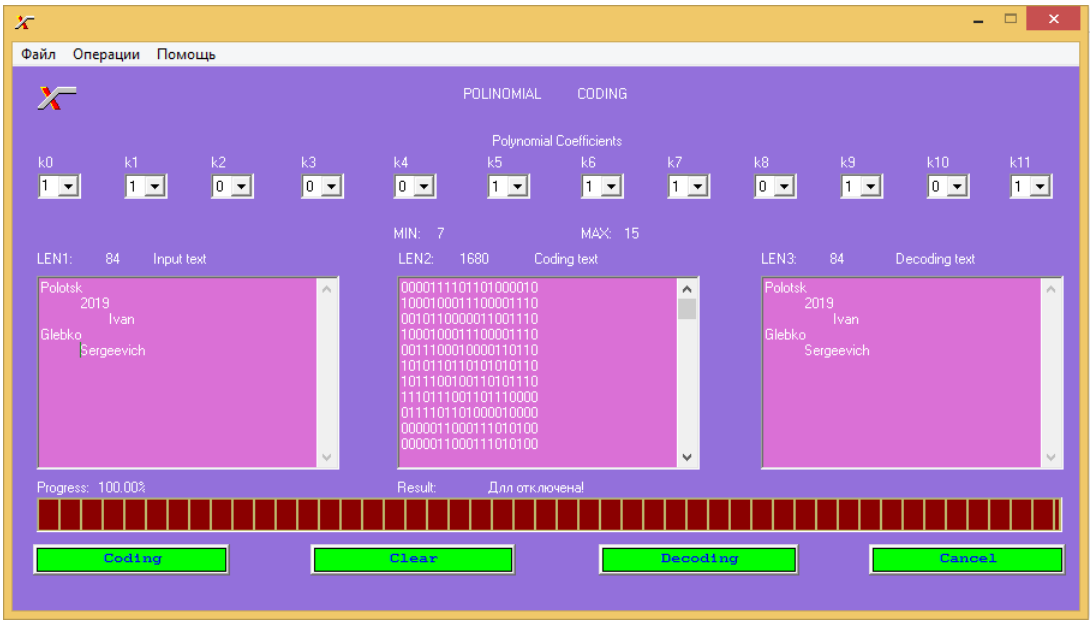

Рис.8. Пример дешифрования фразы каскадом «Polotsk 2019 Ivan Glebko Sergeevich».

Отметим, что использование нелинейных функций для шифрования текста, как и в методе полиномиального кодирования, применяются[2,4]. Однако данный алгоритм используют дополнительно групповые свойства остатков целых чисел по модулю 2. Аналогичные групповые свойства целых остатков по модулю p. частично использует алгоритм шифрования текста эллиптическими кривыми[3].

Совместная программа кодирования-декодирования и подпрограммы для минимального и максимального попарных расстояний между шифрами (малых и больших букв английского алфавита, цифр)

"qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM0123456789"":

```
#include "stdafx.h"
#include<stdio.h>
#include<math.h>
```

```
int sr(int a, int b)
{
  return (a+b)%2;
}
int main()
{
int const mm=34,p0=7,n=11,m=p0, nn=n+p0;
  int i,j,d,p,s,b[p0+1],a,z,h,end[mm+1],gg[mm+1][p0+1],mas[mm+1][nn+1];
int k,kk,shifr[n+m+1],g1[m+2][n+m+2],ost[nn+1],slovo[p0+1];
  int sum,mn[n+m+1],l,ss,ee[nn+1],baza[nn+1],sm[nn+1],chas[nn-n+1];
 char str[mm+2]="Polotsk 2019 Ivan Glebko Sergeevich";

int const mm1=70;
  int b1[p0+1],gg1[mm1+1][p0+1],mas1[mm1+1][nn+1],max,min;
int shifr1[n+m+1],g2[m+2][n+m+2],slovo1[p0+1],dist[mm1+1][mm1+1];
  int ee1[nn+1];
 char
str5[mm1+2]="qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM012345
6789";
mn[0]=1;
mn[1]=1;
mn[2]=0;
mn[3]=0;
mn[4]=0;
mn[5]=1;
mn[6]=1;
mn[7]=1;
mn[8]=0;
mn[9]=1;
mn[10]=0;
mn[11]=1;
 for(i=n+1;i<=n+m;i++)
 {
  mn[i]=0;
 }
  for(i=0;i<=mm;i++)
  {
  printf("sim(%d)=%c(%d)\n",i,str[i],str[i]);
  }
  for(j=0;j<=mm;j++)
  {
  for(i=0;i<=p0;i++)
  {
  gg[j][i]=0;
  }
   }
  for(j=0;j<=mm;j++)
{
a=str[j];
  s=0;
  for (i=0;i<=p0;i++)
  {
```

```
  b[i]=a%2;
  if(a>=2)
  {
                    a=(a-a%2)/2;
                    s=s+1;
                    gg[j][i]=b[i];
                    }
  if(a<=1)
  {
  b[i+1]=a;
  gg[j][i+1]=b[i+1];
            i=p0;
                    }
  }
}
for(j=0;j<=mm;j++)
{
for(i=0;i<=p0;i++)
{
  }
  }
  printf("**\n");
for(d=0;d<=mm;d++)
{
  for(i=0;i<=p0;i++)
  {
            slovo[i]=gg[d][i];
            }
  for(i=n+1;i<=n+m;i++)
  {
            mn[i]=0;
  }
  for(i=0;i<=m;i++)
  for(j=0;j<=n+m;j++)
  {
            g1[i][j]=0;
  }
  for(i=0;i<=m;i++)
  {
  for(j=i;j<=n+m;j++)
  {
g1[i][j]=mn[j-i];
  }
}
  for(i=0;i<=m;i++)
  for(j=0;j<=n+m;j++)
  {
            }
            for(j=0;j<=n+m;j++)
  {
  sum=0;
  for(i=0;i<=m;i++)
```

```
  {
            sum=sum+slovo[i]*g1[i][j];
            sum=sum%2;
  }
            shifr[j] =sum;
            mas[d][j]=shifr[j];
  }
            }
  for(i=0;i<=mm;i++)
  {
  for(j=0;j<=nn;j++)
  {
            if(i==0)
            {
            printf(" shifr(%d %d)=%d\n",i,j,mas[i][j]);
            }
  }
            }
  for(p=0;p<=mm;p++)
{
for(j=0;j<=nn;j++)
{
  ee[j]=mas[p][j];
}

//delenie mnogochlenov
for(i=0;i<=n;i++)
{
            ost[i]=0;
  }
  for(i=0;i<=nn-n;i++)
{
            chas[i]=0;
  }
  for(kk=0;kk<=nn-n;kk++)
{
for(i=0;i<=nn;i++)
{
  baza[i]=0;
}
for(i=0;i<=nn;i++)
{
}
s=0;
for(i=0;i<=nn;i++)
{
  if(ee[nn-i]==0)
  {
  s=s+1;
            }
            else if(ee[nn-i]==1)
            {
```

```
            l=nn-s;
            i=nn;
            }
}
k=l-n;
if(k>=0)
{
chas[k]=1;
  }
if(k<0)
{
for(i=0;i<=n;i++)
{
ost[i]=ee[i];
  }
for(i=0;i<=nn-n;i++)
{
}
kk=nn-n;
}
if(k>=1)
{
for(i=0;i<=nn;i++)
{
baza[i]=0;
}
for(i=0;i<=n;i++)
{
baza[i+k]=mn[i];
}
}
if(k==0)
{
for(i=0;i<=nn;i++)
{
baza[i]=ee[i];
}
for(i=0;i<=n;i++)
{
  sm[i]=sr(mn[i],baza[i]);
  ost[i]=sm[i];
            }
  printf("*******\n");
for(i=0;i<=nn-n;i++)
{
if(chas[i]==1)
{
            }
  else
  {
            chas[i]=0;
                        }
```

```
}
kk=nn-n;
}
for(i=0;i<=nn;i++)
{
sm[i]=sr(baza[i],ee[i]);
}
ss=0;
for(i=0;i<=nn;i++)
{
  ee[i]=sm[i];
  ss=ss+ee[i];
}
if(ss==0)
{
for(i=0;i<=n;i++)
{
  ost[i]=0;
   }
for(j=0;j<=nn-n;j++)
{
if(chas[j]==1)
{
  }
            else
            {
            chas[j]=0;
                              }
  }
kk=nn-n;
}
  }
for(i=0;i<=p0;i++)
{
  gg[p][i]=chas[i];
}
}
for(j=0;j<=mm;j++)
{
z=0;
h=1;
for(i=0;i<=p0;i++)
{
  z=z+gg[j][i]*h;
  h=h*2;
}
  end[j]=z;
}
for(j=0;j<=mm;j++)
{
printf("begin(%d)=%d end(%d)=%c(%d)\n",j,str[j],j,end[j],end[j]);
}
```

```
for(i=n+1;i<=n+m;i++)
 {
  mn[i]=0;
 }
  for(i=0;i<=mm1;i++)
  {

  }
  for(j=0;j<=mm1;j++)
  {
  for(i=0;i<=p0;i++)
  {
  gg1[j][i]=0;
  }
   }
  for(j=0;j<=mm1;j++)
{
a=str5[j];
  s=0;
  for (i=0;i<=p0;i++)
  {
  b1[i]=a%2;
  if(a>=2)
  {
                    a=(a-a%2)/2;
                    s=s+1;
                    gg1[j][i]=b1[i];
                    }
  if(a<=1)
  {
  b1[i+1]=a;
                                        gg1[j][i+1]=b1[i+1];
            i=p0;
                    }
  }
}
for(j=0;j<=mm1;j++)
{

for(i=0;i<=p0;i++)
{
  }
  }

for(d=0;d<=mm1;d++)
{
  for(i=0;i<=p0;i++)

  {
            slovo1[i]=gg1[d][i];
            }
```

```
  for(i=n+1;i<=n+m;i++)
  {
          mn[i]=0;
  }
  for(i=0;i<=m;i++)
  for(j=0;j<=n+m;j++)
  {
          g2[i][j]=0;
  }
          for(i=0;i<=m;i++)
  {
  for(j=i;j<=n+m;j++)
  {
g2[i][j]=mn[j-i];
  }
}
          for(j=0;j<=n+m;j++)
  {
  sum=0;
  for(i=0;i<=m;i++)
  {
          sum=sum+slovo1[i]*g1[i][j];
          sum=sum%2;
  }
          shifr1[j]       =sum;
          mas1[d][j]=shifr1[j];
  }
          }
for(p=0;p<=mm1;p++)
{
for(j=0;j<=nn;j++)
{
  ee1[j]=mas1[p][j];
}
max=-1000;
min=1000;
for(i=0;i<=mm1;i++)
  {
  for(j=0;j<i;j++)
  {
          sum=0;
for(k=0;k<=nn;k++)
{
sum=sum+(mas1[i][k]+mas1[j][k])%2;
}
dist[i][j]=sum;
  }
}
for(i=1;i<=mm1;i++)
  {
  for(j=0;j<i;j++)
  {
```

```
if(max<dist[i][j])
{
  max=dist[i][j];
}
if(min>dist[i][j] && !(i==j)&& dist[i][j]>0 )
{
  min=dist[i][j];
}
  }
}
printf("max=%d\n",max);
printf("min=%d\n",min);
return 0;
}
}
```

Литература

1. Лидовский В.В. Теория информации. Учебное пособие.- М.:2003.-112 С.
2. Пастухов Д.Ф., Пастухов Ю.Ф., Смоляк А.И. Шифрование гиперболическими функциями(http://elib.psu.by:8080/handle/123456789/22089).
3. Пастухов Д.Ф., Пастухов Ю.Ф., Синица П.Р. Шифрование данных на базе эллиптических кривых: учебно-методическое пособие для студентов спец.1-98 01 01 (http://elib.psu.by:8080/handle/123456789/16814).
4. Голубева О.В., Ехилевский С.Г., Пастухов Ю.Ф., Пастухов Д.Ф. Шифрование с помощью нелинейных функций : учебно-методическое пособие для студентов спец.1-98 01 01(http://elib.psu.by:8080/handle/123456789/20320).

УДК 519.72

Кандидат физико-математических наук Пастухов Дмитрий Феликсович

Кандидат физико-математических наук Пастухов Юрий Феликсович

Глебко Иван Сергеевич

Полоцкий государственный университет

2019