Поскольку при выбранном λ выполняется сравнение $u \equiv 1 \bmod n$, определение корректно, так как $L(u) = \dfrac{u-1}{u \bmod n} = \dfrac{u-1 \bmod n}{u \bmod n}$ есть целое число.

Открытым ключом криптосистемы служит пара $k = (n, g)$, а секретным ключом – пара (λ, μ). Для шифрования открытого текста $m \in Z_n$ (множество целых чисел n) выбирается случайное число r, причем $r < n^2$, и вычисляется шифртекст по формуле

$$c = (g^m \times r^n) \bmod n^2.$$

Дешифрование криптограммы c выполняется по формуле

$$m = L(c^\lambda \bmod n^2) \times \mu \bmod n.$$

В рамках данной работы была разработана реализация описанного выше алгоритма в виде модулей на языке Erlang. Erlang – функциональный язык программирования с динамической типизацией, предназначенный для создания распределенных вычислительных систем. Данный язык имеет ряд преимуществ, таких как параллелизм, кроссплатформенность и отказоустойчивость. Поддержка работы со сверхдлинными числами и прозрачность кода сделали Erlang идеальным инструментом для реализации криптосистемы Пэйе.

В настоящее время множество исследователей работают над созданием законченного решения, позволяющего безопасно обрабатывать конфиденциальные данные в облаках. Полностью гомоморфное шифрование способно исключить необходимость хотя бы частичной расшифровки данных для произведения вычислений над ними.

ЛАТИНСКИЙ КВАДРАТ И ЕГО ПРИМЕНЕНИЕ

В. С. Провков, Д. С. Дорохов
(Екатеринбург, УрГУПС, www.usurt.ru)

Латинский квадрат – это квадратная матрица порядка n, каждая строка и каждый столбец которой являются перестановкой элементов конечного множества S, состоящего из n элементов. Наиболее

известным примером латинского квадрата является судоку. Латинские квадраты находят широкое применение в алгебре, комбинаторике, статистике, криптографии, теории кодов и многих других областях.

Впервые в криптографии латинский квадрат был применен в шифре Тритемия, который построил таблицу (соответствующую таблице Кэли группы (Z26; +)) и использовал ее для многоалфавитного шифрования, когда первая буква открытого текста шифруется первым алфавитом (первой строкой таблицы), вторая – вторым и т. д. Впоследствии этот шифр усовершенствовал Дж. Белазо, который придумал пароль. В совокупности с идеей Л. Б. Альберти использовать произвольный алфавит это привело к появлению нового шифра на основе квазигруппы, который стал важной вехой на пути развития криптографии.

Значение латинских квадратов для криптографии иллюстрирует теорема Шеннона, в соответствии с которой единственными совершенными шифрами являются шифры гаммирования, наложение гаммы в которых определяется латинским квадратом.

Также в криптографии существуют нерешенные задачи, связанные с латинскими квадратами, например, недоказанная гипотеза диагоналей, согласно которой у любого латинского квадрата нечетного порядка в каждой строке можно найти диагональ, т. е. неповторяющиеся числа, взятые из разных строк и столбцов.

Поэтому вопрос их генерации является довольно важным.

Задача: написать программу, которая автоматически пишет латинский квадрат, т. е. числа в строках и столбцах не повторяются. Программа была написана нами за 3 дня. Написана она была на языке Python 3.2, и вот ее объяснение.

Для начала нужно подключить библиотеку для функции выбора случайного числа:

```
from random import randint
```

Далее нужно спросить пользователя, сколько строк будет в нашем квадрате. Для этого используем функцию input(), после ввода в переменную num записывается символ и формируется в число.

```
num=input('Enter count of rows : ')
num=int(num)
```

Затем создаем пустой список D=[]. И нужно заполнить его нулями с одинаковым числом строк и столбцов:

```
for n in range(0,num):
  d=[]
  for n in range(0,num):
        d.append(0)
D.append(d)
```

Для числа *n* из чисел от 0 до num присваиваем d=[] – пустой список, потом в этот список добавляем нули, количеством num, после этого в список D добавляем список d, который уже содержит нули, затем начинаем все с начала, делаем это num раз. В итоге получаем (допустим num = 4):

```
[[0,0,0,0], [0,0,0,0], [0,0,0,0], [0,0,0,0]]
```

Создаем пустой список L=[] – это будет тот же список, что и D, только строчки станут столбцами:

```
L=[]
For k in range(0,num):
  l=[]
  for i in range(0,num):
        l.append(D[i][k])
  L.append(l)
```

Для числа *k* из чисел от 0 до num сначала создаем пустой список *l*=[], потом для чисел *I* от 0 до num в *l* записываем число, равное D[*i*][*k*], и затем записываем этот список в новый список L, после повторяем операцию. Таким образом, строки теперь будут равны столбикам и наоборот.

Теперь нужно проверить каждый элемент на наличие такого же элемента в строке и в столбце и, если найдется, изменить его:

```
for i in range(0,num):
  for k in range(0,num):
while D[i].count(D[i][k])>=2 or L[k].count(L[k][i])>=2 or
D[i][k]==0:
```

```
D[i][k]=randint(1,num)
L[k][i]=D[i][k]
```

Для каждого *i* от 0 до num (это нумерация строк) перебираем *k* от 0 до num (нумерация столбцов) и делаем следующее: пока в строке *i* списка D количество символов равному *D*[*i*][*k*] больше или равно 2 или пока в строке *k* списка L количество символов равному L[*k*][*i*] больше или равно 2 или пока символ D[*i*][*k*] равен нулю, нужно присвоить этому символу случайное число в пределах от 1 до num и присвоить символу L[*k*][*i*] то же число.

Выводим полученный список:

```
for n in D:
  print(n)
```

Для каждого *n* из списка D выводим *n* (*n* равно строчке). В итоге при каждом запуске получаются разные списки, являющиеся латинским квадратом, например:

```
[[3, 2, 4, 1],
 [2, 3, 1, 4],
 [4, 1, 2, 3],
 [1, 4, 3, 2]]
```

Если снова запустить программу с таким же количеством строк, то получится абсолютно другой список, но он тоже будет латинским квадратом:

```
[[1, 3, 4, 2],
 [4, 1, 2, 3],
 [2, 4, 3, 1],
 [3, 2, 1, 4]]
```

Итак, программа работает для квадратов 4-го порядка. Однако с увеличением порядка она становится менее стабильна, а к 10 порядку практически перестает работать. Поэтому в дальнейшем мы собираемся усовершенствовать эту программу, написать другую программу и попробовать другие способы генерации, а также заняться решением задачи диагоналей.