Оглавление

Китайская теорема об остатках

С практической точки зрения теорема полезна для выполнения арифметических операций над очень большими числами по, соответственно, большому модулю.

Теорема говорит, что в арифметике по модулю M , если M выражено в виде произведения n целых чисел, являющихся попарно взаимно простыми, то любое целое в $Z_M = \{0,1,...,M-1\}$ можно получить из остатков, образованных этими n числами.

Например, простые множители 10 есть 2 и 5. Теперь возьмем число 9 из Z_{10}. Вычетом по модулю 2 является 1, а вычет по модулю 5 есть 4. Следовательно, в соответствии с теоремой, 9 можно представить как кортеж (1, 4). Почему такое представление удобно с практической точки зрения будет рассмотрено позже.

Пусть M представлено произведением взаимно простых чисел

$$M = \prod_{i=1}^{k} m_i$$

Из этого следует, в частности, что НОД(m_i, m_j)=1, $1 \le i, j \le k$ и $i \ne j$. Например, M =130 можно представить в виде произведения $m_1 = 5$, $m_2 = 26$. Также можно представить в виде произведения $m_1 = 5$, $m_2 = 2$ и $m_3 = 13$.

Теорема позволяет представить любое целое $A \in Z_M$ кортежем

$$A \equiv (a_1, a_2, ..., a_k),$$

где $a_i \in Z_{m_i}$: $a_i = A \bmod m_i, 1 \le i \le k$.

Утверждение 1. Отображение между $A \in Z_M$ и кортежем $(a_1, a_2, ..., a_k)$ биективно. Это означает, в частности, что для каждого целого из Z_M существует уникальный кортеж и наоборот. Более формально, это означает, что существует биективное отображение между Z_M и $Z_{m_1} \times Z_{m_2} \times ... \times Z_{m_k}$.

Утверждение 2. Арифметические операции над числами в Z_M можно эквивалентно представить в виде арифметических операций над кортежами в $Z_{m_1} \times Z_{m_2} \times \ldots \times Z_{m_k}$:

$$\begin{aligned}
(A + B) \bmod M &\Leftrightarrow ((a_1 + b_1) \bmod m_i, \ldots, (a_k + b_k) \bmod m_k) \\
(A - B) \bmod M &\Leftrightarrow ((a_1 - b_1) \bmod m_i, \ldots, (a_k - b_k) \bmod m_k) \\
(A \times B) \bmod M &\Leftrightarrow ((a_1 \times b_1) \bmod m_i, \ldots, (a_k \times b_k) \bmod m_k)
\end{aligned}$$

Чтобы выполнить обратное преобразование, т.е. из кортежа $(a_1, a_2, \ldots, a_k)$ получить соответствующее число A, нужно выполнить следующую последовательность действий:

1. Вычислить $M_i = \frac{1}{m_i}, 1 \le i \le k$. Для этих чисел верно

$$M_i \equiv 0 \pmod{m_i}, \text{ для всех } i \neq j$$

2. Найти M_i^{-1} - обратные значения по умножению для M_i по mod m_i, $1 \le i \le k$. В силу построения M_i обеспечивается существование M_i^{-1} (M_i и m_i взаимно простые).
3. Сформировать числа c_i, $1 \le i \le k$:

$$c_i = M_i \cdot (M_i^{-1} \bmod m_i)$$

4. Получить число A по следующей формуле:

$$A = \left(\sum_{i=1}^{k} a_i \cdot c_i \right) \bmod M \qquad (1)$$

Чтобы убедиться в корректности этой формулы, надо показать, что по рассчитанному таким образом A, можно получить a_i из $A \bmod m_i, 1 \le i \le k$.

Задание 1. Показать, что по рассчитанному по формуле (1) значению A можно получить a_i, если выполнить $A \bmod m_i, 1 \le i \le k$.

Как уже было отмечено, практическая польза применения теоремы проявляется при вычислениях с большими числами (порядка 100-200 десятичных знаков).

Пример.

Пусть $M = 8633 = 89 \times 97 = m_1 \times m_2$. Нужно сложить два числа A=2345 и B=6789 в Z_M.

В соответствии с теоремой, число A=2345 можно представить как $A \Leftrightarrow (a_1, a_2) = (31,17)$, т.к. $2345 \bmod m_1 = 31$, а $2345 \bmod m_2 = 17$. Второе число B=6789 можно представить как $B \Leftrightarrow (b_1, b_2) = (25,96)$. Тогда сложение этих чисел можно выполнить в $Z_{m_1} \times Z_{m_2}$:

$$
\begin{aligned}
(A+B) \bmod M \Leftrightarrow & \big((a_1+b_1) \bmod m_1, (a_2+b_2) \bmod m_2\big) = \\
& \big((31+25) \bmod 89, (17+96) \bmod 97\big) = \\
& \big(56,16\big)
\end{aligned}
$$

Чтобы из $(56,17)$ получить значение $(A+B) \bmod M$ надо:

1. Вычислить $M_1 = \dfrac{M}{m_1} = 97$, $M_2 = \dfrac{M}{m_2} = 89$
2. Найти
$$
\begin{aligned}
M_1^{-1} \bmod m_1 = 78 \\
M_2^{-1} \bmod m_2 = 12
\end{aligned}
$$
3. Сформировать числа c_i:
$$
\begin{aligned}
c_1 = M_1 \cdot (M_1^{-1} \bmod m_1) = 97 \cdot 78 \\
c_2 = M_2 \cdot (M_2^{-1} \bmod m_2) = 89 \cdot 12
\end{aligned}
$$
4. Получить число A:
$$A = \big(a_1 \cdot c_1 + a_2 \cdot c_2\big) \bmod M = (56 \cdot 97 \cdot 78 + 16 \cdot 89 \cdot 12) \bmod 8633 = 501$$

Нетрудно убедиться, что $(A+B) \bmod M = (2345 + 6789) \bmod 8633 = 501$.

Задание 2. Напишите функцию crt(A, m1mk), которая в соответствии с теоремой находит для целого числа $A \in Z_M$ значение в $Z_{m_1} \times Z_{m_2} \times \ldots \times Z_{m_k}$ (рис.1). Здесь m1mk – список, содержащий множители числа $M = m_1 \cdot m_2 \cdot \ldots \cdot m_k$.

```
ma = acrypt.crt(2345, [89, 97])
print('ma={}'.format(ma))
```

```
ma=[31, 17]
```

Рисунок 1

Задание 3. Напишите функцию crt_inv(ma, m1mk), которая из заданного представления числа в $Z_{m_1} \times Z_{m_2} \times \ldots \times Z_{m_k}$ (список ma) рассчитывает соответствующее значение $A \in Z_M$ (рис.2).

```
print('A={}'.format(acrypt.crt_inv(ma, [89, 97])))
```

```
A=2345
```

Рисунок 2

Задание 4. Известно, что в школе учится меньше 1000 учеников. Надо узнать точное количество. Считать всех подряд – нет времени. Поступили таким образом:

1. Построили всех в несколько рядов, в каждом ряду по 5 человек. Оказалось, что последний ряд состоит только из 1 ученика.

2. Построили всех в несколько рядов, в каждом ряду по 8 человек. Оказалось, что последний ряд состоит только из 2 учеников.
3. Построили всех в несколько рядов, в каждом ряду по 19 человек. Оказалось, что последний ряд состоит только из 3 учеников.

   Какое точное количество учеников в школе?

Задание 5. Найдите решение системы

$$\begin{cases} x \equiv 3 \pmod{5}, \\ x \equiv 7 \pmod{9}. \end{cases}$$

Поиск первообразных корней

Задание 6: Написать функцию find_first_primitive_root(n), которая возвращает первый первообразный корень в $(\mathbb{Z}/n\mathbb{Z})^*$. Пример использования показан на рис.3.

Вызов:

```
for p in acrypt.primes(20):
    print('p={}, g={}'.format(p, acrypt.find_first_primitive_root(p)))
```

Результат:

```
p=2, g=1
p=3, g=2
p=5, g=2
p=7, g=3
p=11, g=2
p=13, g=2
p=17, g=3
p=19, g=2
```

Рисунок 3 – Поиск первого первообразного корня в $(\mathbb{Z}/n\mathbb{Z})^*$.

Алгоритм нахождения первообразного корня в $(\mathbb{Z}/p\mathbb{Z})^*$ приведен на рис.4.

1. **if** p=2 then **return** 1
   else g=2
2. Найти простые делители $p_1,\dots,p_r$ числа p-1.
3. **if** для каждого p_i верно, что $g^{\frac{p-1}{p_i}} \not\equiv 1 \pmod{p}$ **then return** g
4. **else** g=g+1, **goto** 3

Рисунок 4 – Алгоритм нахождения первого первообразного корня

По сути, этот алгоритм описан в [1]:

Остановимся теперь на упомянутой выше задаче выбора числа g. При произвольно заданном p она может оказаться трудной задачей, связанной с разложением на простые множители числа $p-1$. Дело в том, что для обеспечения высокой стойкости рассмотренной системы число $p-1$ должно обязательно содержать большой простой множитель (в противном случае алгоритм Полига–Хеллмана, описанный, например, в [28], быстро вычисляет дискретный логарифм). Поэтому часто рекомендуют использовать следующий подход. Простое число p выбирается таким, чтобы выполнялось равенство

$$p = 2q + 1,$$

где q — также простое число. Тогда в качестве g можно взять любое число, для которого справедливы неравенства

$$1 < g < p-1 \quad \text{и} \quad g^q \bmod p \neq 1.$$

Утверждение 3. Пусть p - простое число, $q = \dfrac{p-1}{2}$ - тоже простое число. Тогда, порядок любого элемента из $(Z/pZ)^*$ может принимать значения из набора: 1, 2, q, $p-1$.

Задание 7. Написать функцию find_p_2q_plus_1(bitfield_width), которая возвращает простое число p, такое что $q = \dfrac{p-1}{2}$ - тоже является простым числом. Входной аргумент – размерность простого числа в битах. Пример вызова функции показан на рис.5.

```
p = acrypt.find_p_2q_plus_1(bitfield_width=12)
print('find p={}, is prime (p-1)/2: {}'.format(p, acrypt.is_prime((p-1)//2)))
```

```
find p=5939, is prime (p-1)/2: True
```

Рисунок 5

Так как $p = 2 \cdot q + 1$, то у числа $p - 1$ простыми делителями являются всего два числа $p_1 = 2$ и $p_2 = q$. Тогда алгоритм нахождения первого первообразного корня, приведенный на рис.4, сводится к следующей последовательности действий (рис.6):

1. $p = 2 \cdot q + 1,\ p_1 = 2,\ p_2 = q$
2. **if** $g^{\frac{p-1}{2}} = g^q \not\equiv 1 \pmod{p}$ **and** $g^{\frac{p-1}{q}} = g^2 \not\equiv 1 \pmod{p}$

   then return g
3. **else** g=g+1, **goto** 2

Рисунок 6

Задание 8. Написать функцию find_g(q2_plus_1), которая находит первообразный корень, на основе утверждения 3 (алгоритм на рис.6). Входной аргумент – простое число, найденное с помощью функции find_p_2q_plus_1(). Пример вызова функции показан на рис.7.

```
p = acrypt.find_p_2q_plus_1(bitfield_width=12)
print('find p={}, is prime (p-1)/2: {}'.format(p, acrypt.is_prime((p-1)//2)))
g = acrypt.find_g(p)
print('find g={}'.format(g))
```

```
find p=5399, is prime (p-1)/2: True
find g=7
```

Рисунок 7

Задание 9. Сравнить время поиска первообразных корней с помощью функции find_first_primitive_root() и функции find_g() (рис.8).

```
t0 = time.clock()
p = acrypt.find_p_2q_plus_1(bitfield_width=17)
g = acrypt.find_g(p)
t1 = time.clock()
print('p={}, g={}, time={}'.format(p, g, t1-t0))

t0 = time.clock()
g = acrypt.find_first_primitive_root(p)
t1 = time.clock()
print('p={}, g={}, time={}'.format(p, g, t1-t0))
```

```
p=253679, g=17, time=0.00286413015582343
p=253679, g=17, time=10.042789753507309
```

Рисунок 8 – Сравнение двух функций поиска первого первообразного корня

Задание 10. Сколько первообразных корней по mod p равны 2 для p<100? Т.е. требуется посмотреть все группы $\left(\mathbb{Z}/p_i\mathbb{Z}\right)^*$, $p_i \in \{2,3,5,7,11,13,17,...,97\}$ и определить, в каких из них первообразным корнем является 2.

Дискретный логарифм

Первообразный корень является ключевым понятием дискретного логарифма. Точно так же, как y в выражении $x^y = z$ можно получить через логарифм $y = \log_x z$, так и y в выражении

$$x^y \equiv z \pmod N$$

можно получить через

$$\mathrm{dlog}_{x,N} z = y$$

Для $Z/9Z^*$ первообразным корнем является 2 (рис.9)

$$\begin{array}{lcll}
2^0 & = & 1 & \\
2^1 & = & 2 & \\
2^2 & = & 4 & \\
2^3 & = & 8 & \\
2^4 & \equiv & 7 & \pmod 9 \\
2^5 & \equiv & 5 & \pmod 9 \\
\cdots & \cdots & \cdots & \\
2^6 & \equiv & 1 & \pmod 9 \\
2^7 & \equiv & 2 & \pmod 9 \\
2^8 & \equiv & 4 & \pmod 9 \\
\vdots & & &
\end{array}$$

Рисунок 9

Дискретный логарифм позволяет определить степень первообразного корня (рис.10)

$$\begin{array}{lcl}
dlog_{2,9}\ 1 & = & 0 \\
dlog_{2,9}\ 2 & = & 1 \\
dlog_{2,9}\ 4 & = & 2 \\
dlog_{2,9}\ 8 & = & 3 \\
dlog_{2,9}\ 7 & = & 4 \\
dlog_{2,9}\ 5 & = & 5
\end{array}$$

Рисунок 10

Найденные значения уникальны, если основанием логарифма является первообразный корень, как в рассмотренном примере.

Задание 11. Написать функцию **dlog(g, pub_key, p)**, которая решает задачу дискретного логарифмирования простым перебором. Здесь p - простое число, g - первообразный корень в $(\mathbb{Z}/p\mathbb{Z})^*$, $\text{pub\_key} = g^{\text{priv\_key}} \bmod p$. Функция **dlog** возвращает найденное значение priv_key. Пример использования приведен на рис.11. На рисунке функция find_p_g() выполняет тоже, что и последовательный вызов find_p_2q_plus_1() и find_g().

Вызов:

```
p, g = acrypt.find_p_g(16)
priv_key = p-10
pub_key = pow(g, priv_key, p)
print('p={}, g={}, pub_key={}, priv_key={}'.format(p, g, pub_key, priv_key))
t0 = time.clock()
priv_key = acrypt.dlog(g, pub_key, p)
t1 = time.clock()
dt = t1-t0
print('find priv_key={}, time={}'.format(priv_key, dt))
```

Результат:

```
p=75707, g=2, pub_key=20849, priv_key=75697
find priv_key=75697, time=0.4101172076339311 7
```

Рисунок 11

Задание 12. Постройте график (рис.12). График показывает, сколько времени на конкретном ПК требуется для решения задачи дискретного логарифмирования в $(Z_p)^*$ простым перебором. По оси абсцисс – размерность простого числа p в битах. Простое число получено с помощью функции find_p_2q_plus_1(). По оси ординат – время в секундах, затраченное на вычисление dlog(g, p-3, p). Первообразный корень g получен с помощью функции find_g().

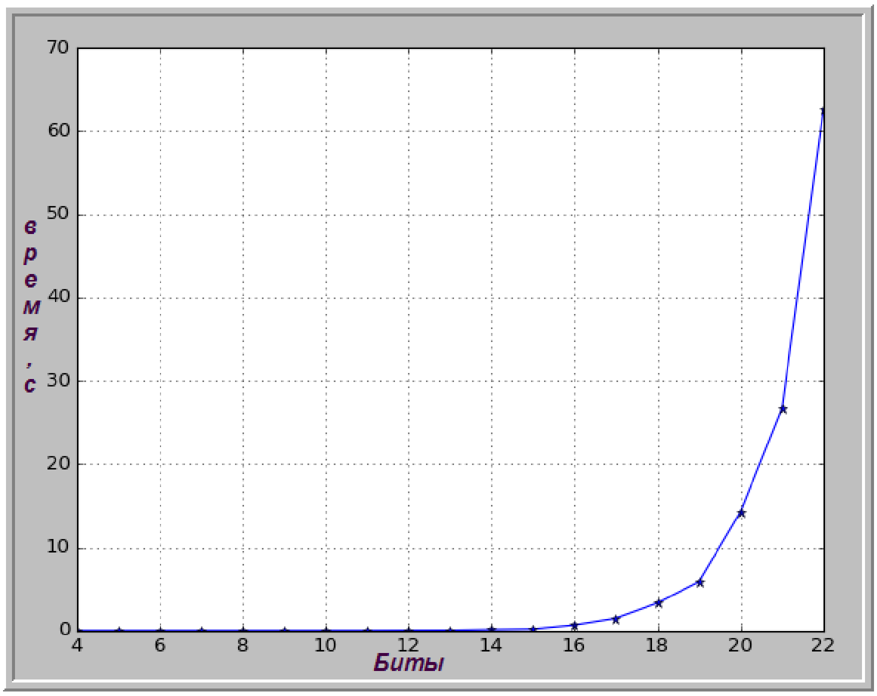


Рисунок 12

Задание 13. Найдите $a \in (Z/13Z)^{*}$, для которых не существуют значений $\mathrm{dlog}_{3,13} a$.

ElGamal

Задание 14. Создать текстовый файл ‘fio.txt’. Записать в него свою фамилию, имя и отчество. Извлечь содержимое файла в список data с помощью функции read_data_1byte. Преобразовать этот список в массив целых чисел nums размерностью в три байта. Все числа из этого списка нужно зашифровать с помощью алгоритма ElGamal. Прежде всего, нужно выбрать параметры этого шифра: p, g, секретный ключ priv_key, открытый ключ pub_key. Так как p должно быть больше любого шифруемого числа, то надо найти максимальное число из массива nums. Допустим, максимальное значение будет занесено в переменную m. Теперь надо определить, сколько бит требуется для записи этого числа:

```
bitfield_width = math.floor(math.log2(m))
```

Так как, параметр p должен быть больше m, то под битовое представление числа p нужно выделить большее количество бит:

```
bitfield_width = bitfield_width + 2
```

Теперь с помощью функции **find_p_g(bitfield_width)** получить простое число p и соответствующий первообразный корень g.

Сформировать произвольный секретный ключ, например:

```
priv_key = 4356
```

Найти открытый ключ pub_key:

```
pub_key = pow(g, priv_key, p)
```

Зашифровать с помощью функции **elgamal_encrypt(pub_key, g, p, m)** все числа из массива nums. Зашифрованные числа сохранить в массиве encrypt_nums. В этом массиве будет ровно в два раза больше чисел, чем в nums.

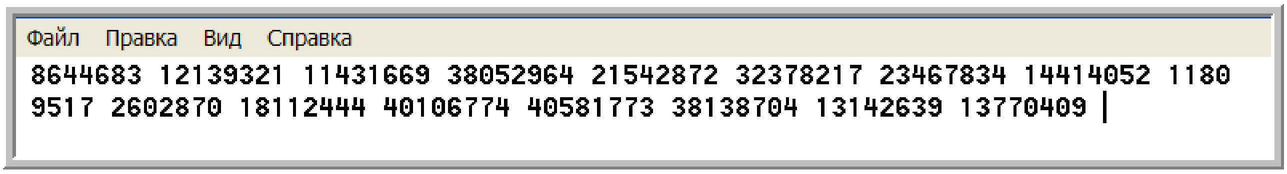
Вставить функцию write_numbers() в модуль read_write_file:

```
def write_numbers(fname, numbers):
    fo = open(fname, 'w')
    for i in numbers:
        fo.write(str(i)+' ')
    fo.close()
```

Записать содержимое массива encrypt_nums в файл ‘encrypt_file.txt’:

```
read_write_file.write_numbers('encrypt_file.txt', encrypt_nums)
```

Содержимое файла можно просмотреть в обычном «Блокноте», например:



В этом файле содержится зашифрованное сообщение из исходного файла ‘fio.txt’. Теперь нужно расшифровать содержимое зашифрованного файла ‘encrypt_file.txt’. Для этого, вставить функцию **read_numbers(fname)** в модуль read_write_file:

```
def read_numbers(fname):
    cypher_data = read_data_1byte(fname)
    # form string
    s = ''
    for i in cypher_data:
        s += (str(chr(i)))
    s = s.split()
    # form list of large numbers
    encrypt_nums = []
    for i in s:
        encrypt_nums.append(int(i))
    return encrypt_nums
```

Прочитать содержимое зашифрованного файла:

```
encrypt_nums = read_write_file.read_numbers('encrypt_file.txt')
```

Теперь в массиве encrypt_nums содержатся те же самые числа, которые можно было видеть в файле с помощью «Блокнота»:

```
encrypt_nums = [8644683, 12139321, 11431669, 38052964, 21542872, 32378217, 23467834, 14414052, 11809517, 2602870,
 18112444, 40106774, 40581773, 38138704, 13142639, 13770409]
```

С помощью функции **elgamal_decrypt(pri_key, p, c1, c2)** расшифровать числа из массива encrypt_nums. Результат сохранить в массив decrypt_nums. Преобразовать расшифрованные числа в последовательность байт data1 с помощью функции **get_data_from_blocks().** Записать полученную последовательность байт в файл ‘fio1.txt’ с помощью функции:

```
read_write_file.write_data_1byte('fio1.txt', data1)
```

В файле 'fio1.txt' должна быть ваша фамилия, имя и отчество.

Задание 15.

Расшифровать файл b4_ElG_c.png. Известны следующие параметры: p = 9887455967, g = 5, pub_key= 3359661584, priv_key= 543,block_size=4, длина исходных данных 24776 байт.

RSA

На рис.13 приведена схема, которая отражает процесс шифрования данных с помощью криптографии с открытым ключом.

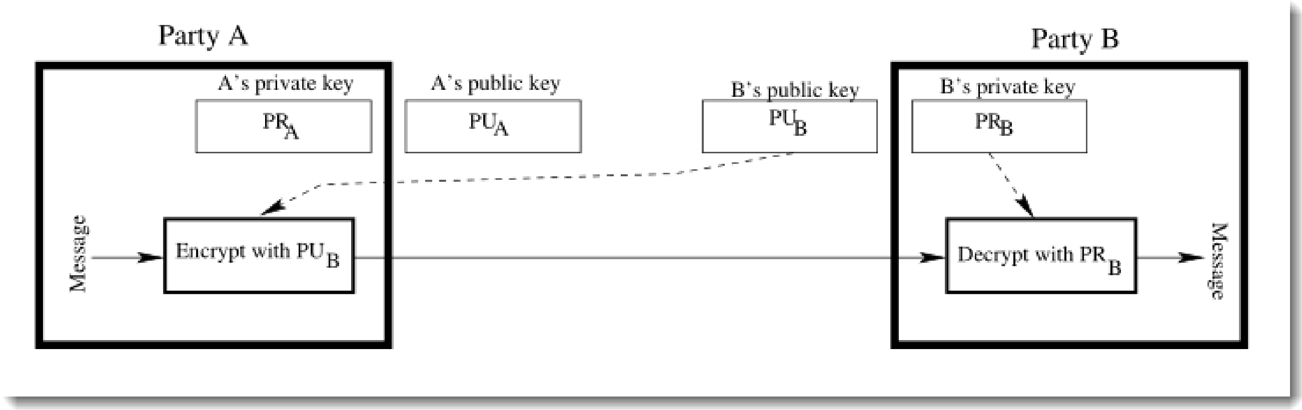


Рисунок 13

Ниже (рис. 14) приведены шаги для использования в схеме на рис.13 алгоритма RSA.

Bob	Alice
Key Creation	
Choose secret primes p and q. Choose encryption exponent e with $\gcd(e, (p-1)(q-1)) = 1$. Publish $N = pq$ and e.	
Encryption	
	Choose plaintext m. Use Bob's public key (N, e) to compute $c \equiv m^e \pmod{N}$. Send ciphertext c to Bob.
Decryption	
Compute d satisfying $ed \equiv 1 \pmod{(p-1)(q-1)}$. Compute $m' \equiv c^d \pmod{N}$. Then m' equals the plaintext m.	

Рисунок 14

Задание 16. Создать текстовый файл ‘fio.txt’. Записать в него свою фамилию, имя и отчество. Извлечь содержимое файла в список data с помощью функции read_data_1byte. Преобразовать этот список в массив целых чисел nums размерностью в три байта. Все числа из этого списка нужно зашифровать с помощью алгоритма RSA.

Прежде всего, нужно выбрать параметры этого шифра: p, q, n, секретный ключ priv_key, открытый ключ pub_key. Так как n должно быть больше любого шифруемого числа, то надо найти максимальное число из массива nums. Определить, сколько бит нужно выделить под это число. Определить, сколько бит нужно для числа n. Сгенерировать нужной размерности числа p, q, n.

Выбрать открытый ключ pub_key. Для этого надо сгенерировать простое число e и проверить, что оно удовлетворяет условию: $\gcd(e,(p-1)\cdot(q-1))=1$. Это означает, что должно выполняться также: $\gcd(e,(p-1))=1$ и $\gcd(e,(q-1))=1$. Для проверки последних двух условий достаточно убедиться, что $q \bmod e \neq 1$ и $p \bmod e \neq 1$ или в Питоне:

```
q % e != 1 and p % e != 1
```

Найти секретный ключ priv_key. Для этого нужно найти такое d, при котором выполняется $ed \equiv 1\,(\bmod\,(p-1)(q-1))$, где e - открытый ключ:

```
fi_n = (p-1)*(q-1)
priv_key = acrypt.findModInverse(pub_key, fi_n)
```

Написать функции:

```
def rsa_encrypt(m, pub_key, n):
#  pub_key = public key
# m = number < n
```

```
def rsa_decrypt(c, priv_key, n):
# priv_key = private key
# c = ciphertext
```

Зашифровать с помощью функции **rsa_encrypt(m, pub_key, n)** все числа из массива nums. Зашифрованные числа сохранить в массиве encrypt_nums. Записать содержимое массива encrypt_nums в файл ‘encrypt_file.txt’ (функция **write_numbers**).

В этом файле содержится зашифрованное сообщение из исходного файла ‘fio.txt’. Теперь нужно расшифровать содержимое зашифрованного файла ‘encrypt_file.txt’. Для этого надо прочитать содержимое зашифрованного файла обратно в массив encrypt_nums (функция **read_numbers**).

С помощью функции **rsa_decrypt(c, priv_key, n)** расшифровать числа из массива encrypt_nums. Результат сохранить в массив decrypt_nums. Преобразовать расшифрованные числа в последовательность байт data1 с помощью функции **get_data_from_blocks().** Записать полученную последовательность байт в файл ‘fio1.txt’ (функция **write_data_1byte**). В файле 'fio1.txt' должна быть ваша фамилия, имя и отчество.

Задание 17. Расшифровать файл ddd10_rsa_c.bmp. Известны следующие параметры: p= 7919, q= 6599, pub_key= 2011, priv_key= 17457619, len_data= 37451, block_size=3.

Задание 18. RSA шифртекст. Дешифровать, если известно, что n=18923, e=1261. Преобразование текста в числа осуществлялось по три символа по следующей схеме:

$$\begin{array}{lcccr} DOG & \to & 3\times 26^2 + 14\times 26 + 6 & = & 2398 \\ CAT & \to & 2\times 26^2 + 0\times 26 + 19 & = & 1371 \\ ZZZ & \to & 25\times 26^2 + 25\times 26 + 25 & = & 17575. \end{array}$$

шифртекст = {12423, 11524, 7243, 7459, 14303, 6127, 10964, 16399,
9792, 13629, 14407, 18817, 18830, 13556, 3159, 16647,
5300, 13951, 81, 8986, 8007, 13167, 10022, 17213,
2264, 961, 17459, 4101, 2999, 14569, 17183, 15827,
12693, 9553, 18194, 3830, 2664, 13998, 12501, 18873,

12161, 13071, 16900, 7233, 8270, 17086, 9792, 14266,
13236, 5300, 13951, 8850, 12129, 6091, 18110, 3332,
15061, 12347, 7817, 7946, 11675, 13924, 13892, 18031,
2620, 6276, 8500, 201, 8850, 11178, 16477, 10161,
3533, 13842, 7537, 12259, 18110, 44, 2364, 15570,
3460, 9886, 8687, 4481, 11231, 7547, 11383, 17910,
12867, 13203, 5102, 4742, 5053, 15407, 2976, 9330,
12192, 56, 2471, 15334, 841, 13995, 17592, 13297,
2430, 9741, 11675, 424, 6686, 738, 13874, 8168,
7913, 6246, 14301, 1144, 9056, 15967, 7328, 13203,
796, 195, 9872, 16979, 15404, 14130, 9105, 2001,
9792, 14251, 1498, 11296, 1105, 4502, 16979, 1105,
56, 4118, 11302, 5988, 3363, 15827, 6928, 4191,
4277, 10617, 874, 13211, 11821, 3090, 18110, 44,
2364, 15570, 3460, 9886, 9988, 3798, 1158, 9872,
16979, 15404, 6127, 9872, 3652, 14838, 7437, 2540,
1367, 2512, 14407, 5053, 1521, 297, 10935, 17137,
2186, 9433, 13293, 7555, 13618, 13000, 6490, 5310,
18676, 4782, 11374, 446, 4165, 11634, 3846, 14611,
2364, 6789, 11634, 4493, 4063, 4576, 17955, 7965,
11748, 14616, 11453, 17666, 925, 56, 4118, 18031,
9522, 14838, 7437, 3880, 11476, 8305, 5102, 2999,
18628, 14326, 9175, 9061, 650, 18110, 8720, 15404,
2951, 722, 15334, 841, 15610, 2443, 11056, 2186};

Литература

[1] Рябко Б.Я., Фионов А.Н., Криптографические методы защиты информации: учебное пособие для вузов. – М.: Горячая линия-Телеком, 2005. – 229 с.

[2] Черемушкин А.В. Лекции по арифметическим алгоритмам в криптографии. – М.: МЦНМО, 2002. – 104 с.

[3] Stallings W, “Cryptography And Network Security. Principles And Practice”, 5th Edition, 2011.