Оглавление

[Задание 1 2](#_Toc498156864)

[Задание 2 4](#_Toc498156865)

[Задание 3 5](#_Toc498156866)

[Задание 4 5](#_Toc498156867)

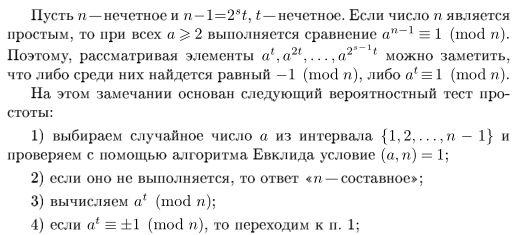
[Задание 5 8](#_Toc498156868)

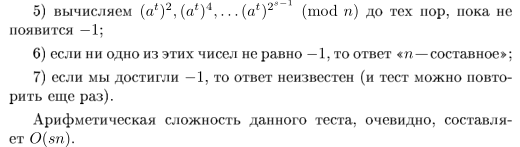
[Задание 6 9](#_Toc498156869)

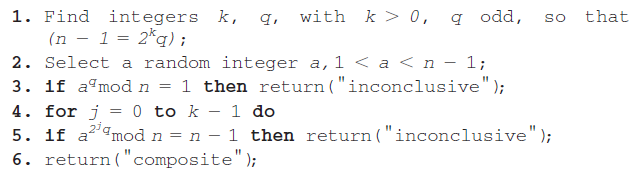
[Задание 7 10](#_Toc498156870)

# Задание 1

Вероятностный тест проверки чисел на простоту – тест Рабина-Миллера [2]





Алгоритм:

Реализация:

|  |
| --- |
| def rabin\_miller(n):  # Returns True if num is a prime number.  q = n - 1  k = 0  while q % 2 == 0:  # keep halving s until it is even (and use t  # to count how many times we halve s)  q = q // 2  k += 1  t = 5  for trials in range(t): # try to falsify num's primality 5 times  a = random.randrange(2, n - 1)  v = pow(a, q, n)  if v != 1: # this test does not apply if v is 1.  i = 0  while v != (n - 1):  if i == k - 1:  return False, 0  else:  i = i + 1  v = (v \*\* 2) % n  probability\_of\_prime = 1 - 1.0/(4 \*\* t)  return True, probability\_of\_prime |

а) Сгенерировать «свое» нечетное число n из интервала [501, 2002) (2002-не включается):

|  |
| --- |
| n = random.randrange(501, 2002)  while n % 2 == 0:  n = random.randrange(501, 2002)  print('n={}'.format(n)) |

б) Продемонстрировать пошаговую работу алгоритма с полученным числом (для этого использовать отладчик или печать значения с помощью print()) для двух случаев: когда число простое и когда число составное.

Пример вызова функции показан ниже:

|  |
| --- |
| n = 1753  isprime, p = rabin\_miller(n)  if isprime:  print('{} is prime number with probability {}'.format(n, p))  else:  print('{} is composite'.format(n)) |

в) Поставьте прохождение теста только 4 раза. Для n=221 сделайте так, чтобы параметр а последовательно принимал значения: 21, 47, 174, 200. Является ли число 221 простым согласно этому алгоритму при таких условиях?

г) Функция **is\_prime**(n), которая на основе теста (функция rabin\_miller()) определяет, является ли любое целое число n простым или нет:

|  |
| --- |
| def is\_prime(n):  # Return True if n is a prime number. This function does a quicker  # prime number check before calling rabin\_miller().  if n < 2:  return False # 0, 1, and negative numbers are not prime  # About 1/3 of the time we can quickly determine if n is not prime  # by dividing by the first few dozen prime numbers. This is quicker  # than rabin\_miller().  low\_primes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907, 911, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983, 991, 997]  if n in low\_primes:  return True  # See if any of the low prime numbers can divide n  for prime in low\_primes:  if n % prime == 0:  return False  # If all else fails, call rabin\_miller() to determine if n is a prime.  return rabin\_miller(n) |

Найдите с помощью этой функции 10 простых чисел из диапазона [13000, 14000].

д) Проверьте, что следующие два последовательные нечетные числа – простые: 1000000000061, 1000000000063

е) Сколько простых чисел в последовательности: 1001! + 2, 1001! + 3, ... , 1001! + 1000, 1001! + 1001?

# Задание 2

1. Функция π(x) – возвращает количество простых чисел, меньших или равных x. Написать функцию pi\_function:

print(pi\_function(24))

>>9

1. Дописать значения π(x) и x/ln(x) в таблицу 1 для х = 105 (π(x)-количество простых чисел, меньших или равных x, x/ln(x)-оценка значения π(x)):

Таблица 1

|  |
| --- |
|  |

# Задание 3

Написать функцию **generate\_large\_prime(bitfield\_width).** Функция возвращает простое число, которое в двоичной системе счисления содержит ровно указанное количество бит (**bitfield\_width**).

Решение:

Организовать в этой функции бесконечный цикл. В этом цикле сгенерировать число-кандидат:

|  |
| --- |
| candidate = random.getrandbits(bitfield\_width) |

Гарантировать нечетность полученного числа:

|  |
| --- |
| if candidate & 1 == 0:  candidate += 1 |

Гарантировать, что два старших бита выставлены в 1:

|  |
| --- |
| candidate |= (1 << bitfield\_width - 1)  candidate |= (2 << bitfield\_width - 3) |

Вызвать функцию is\_prime().

Пример вызова функции показан на рис.1. Результат вызова представлен на рис. 2.

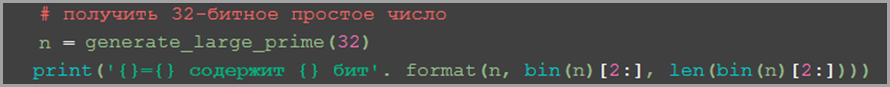


Рис. 1. Вызов функции generate\_large\_prime(32)

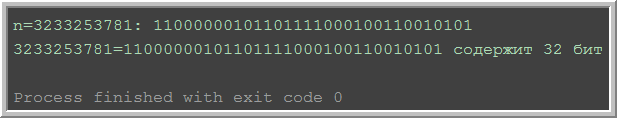


Рис. 2. Результат вызова функции generate\_large\_prime(32)

Получить простое число размерностью 41 бит, 53 бита и 120 бит.

# Задание 4

Преобразование блока сообщения в большое целое число показано на рис. 3.

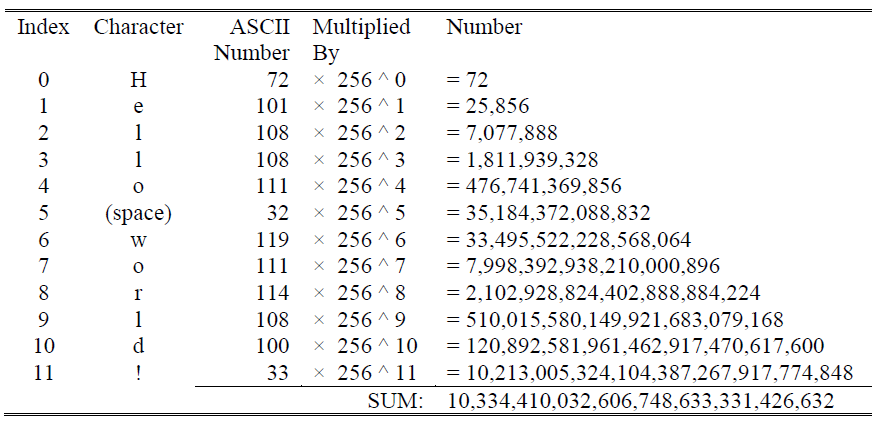


Рис. 3.

Реализация в виде функции **getBlocksFromText**(message, block\_size):

|  |
| --- |
| def getBlocksFromText(message, block\_size):  # Converts a string message to a list of block integers. Each integer  # represents block\_size string characters.  message\_bytes = message.encode('ascii') # convert the string to bytes  block\_ints = []  for blockStart in range(0, len(message\_bytes), block\_size):  # Calculate the block integer for this block of text  block\_int = 0  for i in range(blockStart, min(blockStart + block\_size, len(message\_bytes))):  block\_int += message\_bytes[i] \* (256 \*\* (i % block\_size))  block\_ints.append(block\_int)  return block\_ints |

Показать, что для сообщения ‘Hello world!’ и размера блока 12 полученное с помощью функции **getBlocksFromText** число соответствует значению, приведенному на рис. 3:

|  |
| --- |
| num = acrypt.getBlocksFromText(message='Hello world!', block\_size=12)  print('message is a number: {}'.format(num)) |

Привести результат, который возвращает эта функция, когда размер блока установлен в значение 4.

На основе этой функции написать ее модификацию (функцию **get\_blocks\_from\_data**). Суть: вместо строки, на вход функции подается список байт. Пример вызова этой функции:

|  |
| --- |
| data = [72, 101, 108, 108, 111, 32, 119, 111, 114, 108, 100, 33]  num = get\_blocks\_from\_data(data, block\_size=12)  print('message is a number: {}'.format(num)) |

Результат сравните с рис. 3.

# Задание 5

Функция обратного преобразования **getTextFromBlocks**:

|  |
| --- |
| def getTextFromBlocks(block\_ints, message\_length, block\_size):  # Converts a list of block integers to the original message string.  # The original message length is needed to properly convert the last  # block integer.  message = []  for block\_int in block\_ints:  block\_message = []  for i in range(block\_size - 1, -1, -1):  if len(message) + i < message\_length:  # Decode the message string for the 128 (or whatever  # blockSize is set to) characters from this block integer.  ascii\_number = block\_int // (256 \*\* i)  block\_int %= 256 \*\* i  block\_message.insert(0, chr(ascii\_number))  message.extend(block\_message)  return ''.join(message) |

Восстановить сообщение из числа: 10334410032606748633331426632

|  |
| --- |
| msg = getTextFromBlocks(block\_ints=num, message\_length=12, block\_size=12)  print('message is {}'.format(msg)) |

Написать функцию **get\_data\_from\_blocks().** Суть: вместо строки, функция возвращает список байт. Достаточно убрать функции chr и join. Пример вызова этой функции:

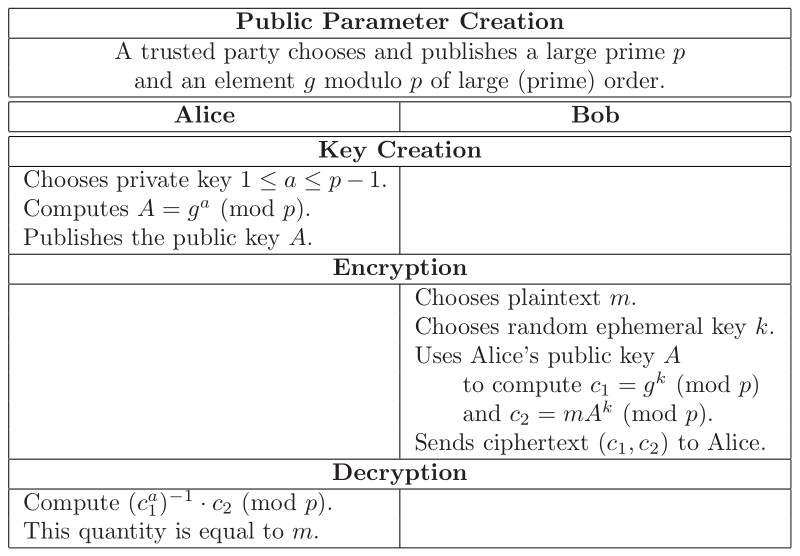
|  |
| --- |
| data1 = get\_data\_from\_blocks(block\_ints=num, message\_length=12, block\_size=12)  print('message is {}'.format(data1)) |

# Задание 6

Создать текстовый файл. Записать в него свою ФИО (полностью). Извлечь содержимое файла в список data. Преобразовать этот список в массив целых чисел размерностью в три байта. Выполнить обратное преобразование. Результат сохранить в новом файле.

# Задание 7

Криптосистема ElGamal:

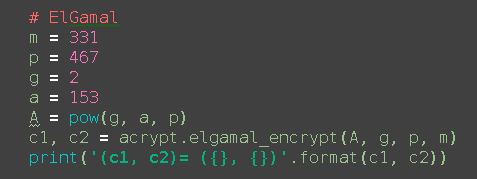


Напишите функции:

|  |
| --- |
| def elgamal\_encrypt(pub\_key, g, p, m):  # pub\_key = public key  # g = generator  # p = prime  # message = number < p |

|  |
| --- |
| def elgamal\_decrypt(pri\_key, p, c1, c2):  # pri\_key = private key  # p = prime  # (с1 ,с2) = ciphertext |

Пример шифрования:



Результат:



Пример расшифрования:



Результат:



Проверьте, что все работает при случайном задании к:

|  |
| --- |
| k = random.randint(1,p-1) |

Известно: , g=7, pub\_key= 833287206, (с1,с2) = (1457850878, 2110264777).

Найти: m

Зашифровать ФИО