Министерство образования и науки Российской Федерации  
ФГАОУ ВПО «Уральский федеральный университет имени первого  
Президента России Б.Н. Ельцина»

Научно-учебный центр «Информационная Безопасность»

Отчет

По дисциплине «Криптографические методы защиты информации»

Лабораторная работа №5

Подпись ФИО

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.В. Агафонов

Студенты \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Л.В. Родионов

И.А. Ибрагимов

Группа РИ-481220

Екатеринбург 2022

Целью данной лабораторной работы является изучение принципов ассиметричной криптографии на примере реализации шифросистемы RSA

**Ход работы:**

**2.Создать программу для генерации пар ключей <n,e> и <n,p,q,d> для шифросистемы RSA (с. 311 пособия). При этом числа p и q выбирать случайным образом из сформированной последовательности из ста простых чисел, начиная с 17.**

Код сценария с комментариями приведен ниже:

# Генерация ключей  
f\_open\_key = open('open\_key.txt', 'w')  
f\_close\_key = open('close\_key.txt', 'w')  
primes = [] # В этот массив запишем первые 100 простых чисел, начиная с 17  
i = 0  
k = 17  
while i != 100: # пока не набрано 100 чисел  
 if sympy.isprime(k): # Проверяем простое ли число  
 primes.append(k) # Если простое то добавляем в массив  
 i += 1  
 k += 1  
 else: # Иначе проверяем следующее число  
 k += 1  
num = random.randint(0, len(primes) - 2) # Из получившихся чисел выбираем случайное и присваиваем p  
p = primes[num]  
q = primes[num + 1] # Присваиваем q значение следующего простого числа после случайно выбранного  
n = p \* q # вычисляем n и фи  
phi = (p - 1) \* (q - 1)  
delit = [] # Массив делителей числа фи  
for i in range(2, phi): # Перебираем множители  
 if sympy.gcd(i, phi) == 1 and sympy.isprime(i): # Если НОД фи и множителя равен 1, добавляем в массив  
 delit.append(i)  
e = delit[  
 random.randint(0, len(delit) - 1)] # Выбираем e случайным взаимнопростым числом из сформированного списка  
d = 0  
u = 1  
while d == 0: # Ищем d из условия d \* e - u \* phi = 1  
 if (phi \* u + 1) % e == 0:  
 d = int((phi \* u + 1) / e)  
 else:  
 u += 1  
f\_open\_key.write(bin(n) + bin(e)) # записываем в файл открытый ключ в бинарном формате  
f\_close\_key.write(bin(n) + bin(d)) # записываем закрытый ключ в бинарном формате в файл  
f\_open\_key.close()  
f\_close\_key.close()

В рамках учебных целей выбираем p и q относительно небольшими простыми числами, находящимися недалеко друг от друга.

Программа записывает в массив primes проверяемое число, если функция sympy.isprime() возвращает True. Далее из массива primes случайным образом выбираются числа p и q.

Далее находятся числа n=p\*q и f(n) = (p-1)\*(q-1). Далее следует блок поиска взаимно-простых чисел для f(n). Начиная с двойки проверяются все числа до f(n)-1 включительно. Если НОД проверяемого числа и f(n) равен 1, то числа взаимно простые и проверяемое число записывается в массив. Далее из полученного списка случайным образом выбирается число и присваивается переменной **e**.

Поиск d – элемента закрытого ключа производится следующим образом: e \* d = 1 mod f(n). Значит d = (1 + f(n) \* u)/e. Подбирая значение **u** находим значение d. Открытый ключ, составляющий <n,e> записываем в файл open\_key\_file.txt а закрытый ключ <d,n> в файл close\_key\_file.txt

**3.Создать программу, которая зашифровывает и расшифровывает файлы с использованием шифросистемы RSA с использованием ранее сформированных ключей.**

Созданная программа позволяет выбрать режим работы.

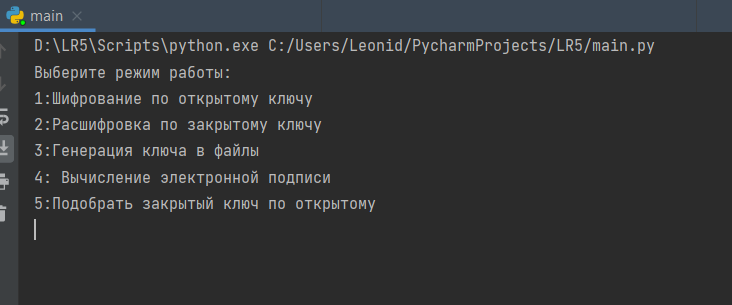


Рисунок – Режимы работы программы

Для шифрования файлов была выбрана шифровать каждый байт исходного текста. Без всяких сомнений такой способ открывает возможность успешного проведения криптоанализа, в случае, когда рассматривается структура шифротекста, однако для учебных целей такая работа блока шифрования обусловлена тем, что в качестве чисел p и q все равно выступают довольно маленькие простые числа, подобрать закрытый ключ в таком случае не составляет больших проблем, если злоумышленник владеет открытым ключом. В уязвимости рассматриваемого в данной работе алгоритма упомянем в следующих пунктах.

Байты открытого текста поступают в блок шифрования. Если поступающий байт равен Mi, а его зашифрованный аналог Ci, то преобразование Mi к Ci выглядит так:

Ci = M^e mod n

Далее байты переводятся в биты с указанием начала следующего байта и записываются в файл, имя которого было указано пользователем в начале работы блока шифрования

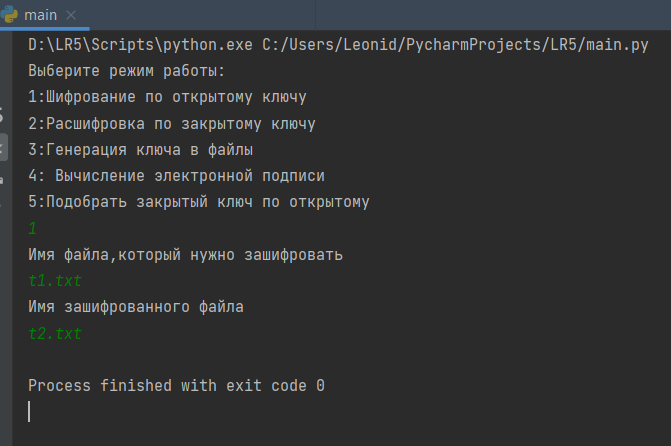


Рисунок – Работа в режиме шифрования

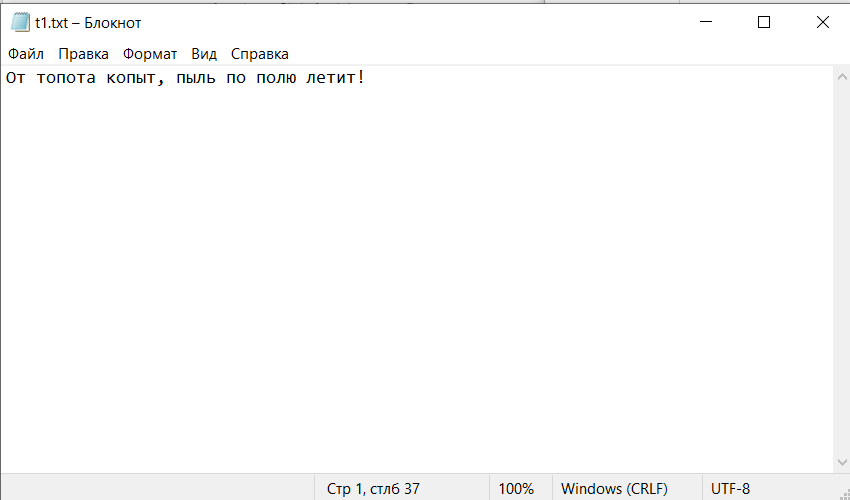


Рисунок – Открытый текст

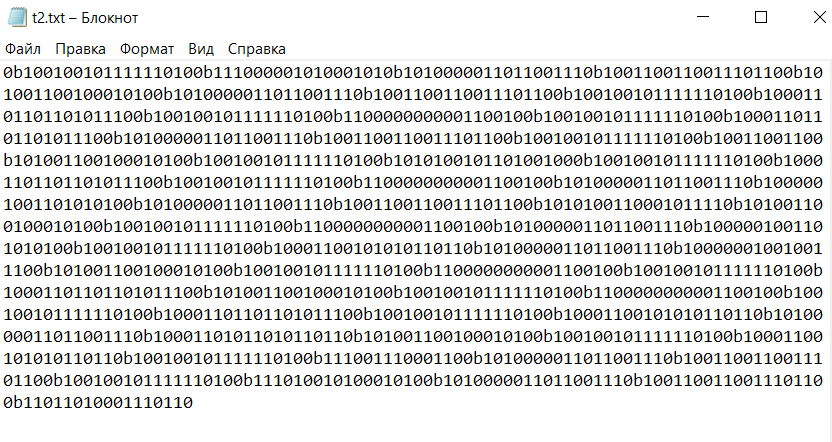


Рисунок – Шифротекст

0b – означает начало следующего числа

Запись в файл, можно было производить побайтово, однако для учебных целей мы решили пропустить этот этап, поскольку увеличивается сложность алгоритмов шифрования и дешифрования, следовательно увеличивается время работы программы.

Блок расшифрования работает следующим образом.:

Считываются двоичные записи чисел, превращаются в int, далее эти числа с помощью выражения M = C^d mod n преобразуются к начальным байтам открытого текста, которые в итоге составляют расшифрованный файл.

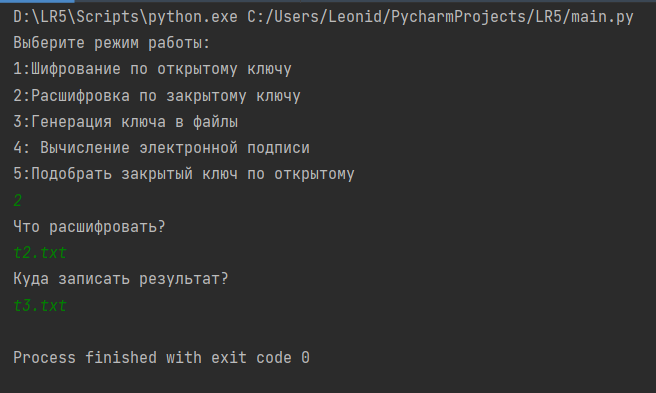


Рисунок – Работа сценария в режиме расшифровки

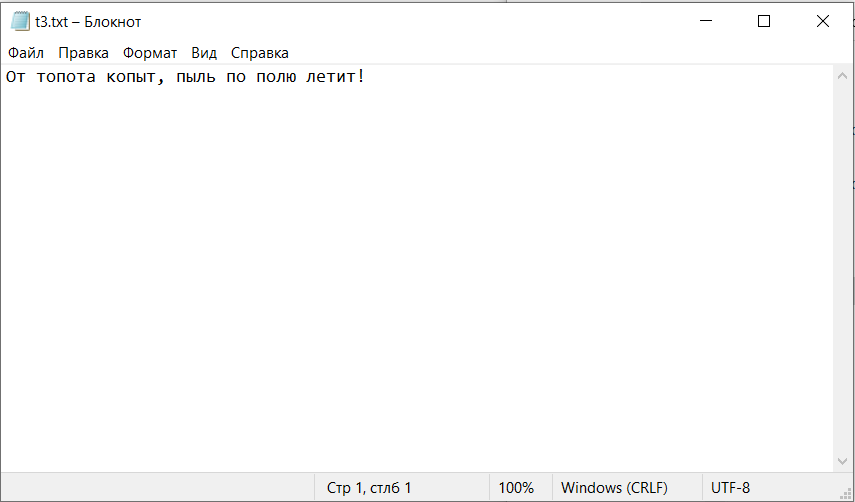


Рисунок – Расшифрованный файл

**4.Зашифровать файл с помощью открытого ключа и затем расшифровать с помощью закрытого. Рассчитать контрольную сумму файла (для учебных целей можно использовать сложение по модулю 2 всех байтов файла) и зашифровать ее закрытым ключом, получив электронную подпись (ЭП) файла, которую необходимо сохранить в отдельный файл.**

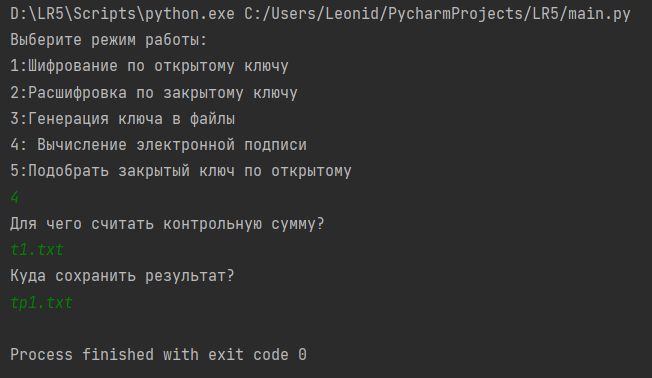


Рисунок – Режим вычисления цифровой подписи

Вычисление электронной подписи происходит в два этапа:

1. Вычисление контрольной суммы файла, путем сложения по модулю 2 всех его байтов
2. Шифрование полученного числа по правилу C = M ^ d mod n
3. Запись числа в файл

Убедимся в правильности работы: вычислим электронную подпись для файла t1.txt и t3.txt из предыдущего пункта. Поскольку содержимое фалов одинаковое, Электронные подписи должны совпадать.

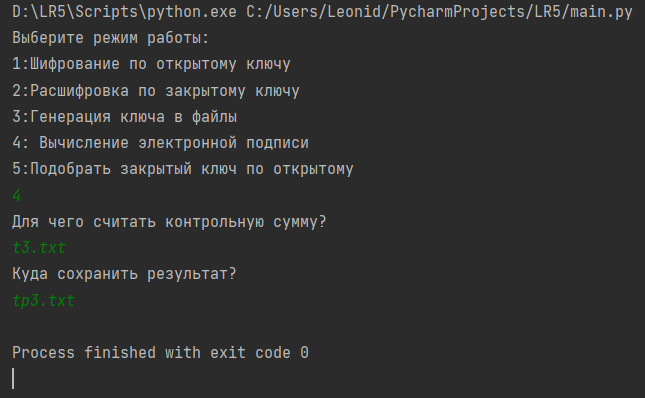


Рисунок – Вычисление цифровой подписи для расшифрованного файла t3.txt

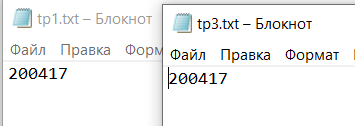


Рисунок – Электронные подписи файлов с одинаковым содержимым

**5.Теперь попробуем обмануть программу вычисления ЭП.** Поскольку механизм вычисления зависит от байтов, составляющих файл, можно поменять буквы местами, или внеся изменения, записать все символы еще раз в любом порядке:

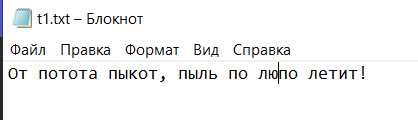


Рисунок – Изменение порядка букв

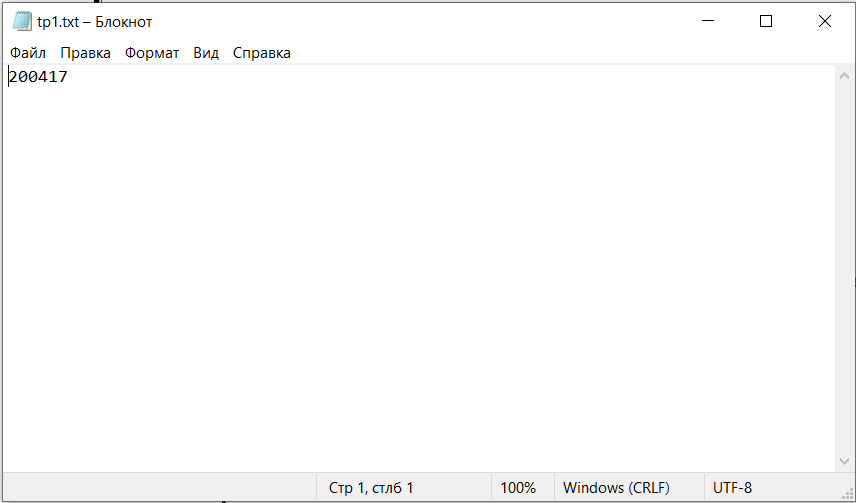


Рисунок – ЭП не изменилась от порядка букв

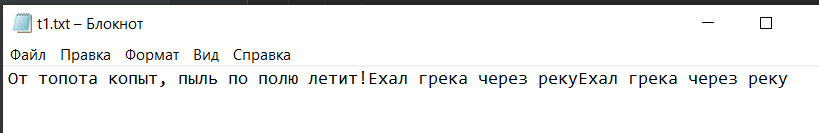


Рисунок – Дважды добавили лишний текст

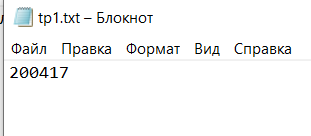


Рисунок – ЭП не изменилась

Для варианта, когда требуется удалить часть текста или заменить ее, требуется отдельно найти ЭП этой части текста и скомпенсировать ее в файле, тогда финальная ЭП не изменится

**6.Зная о небольшом диапазоне значений используемых p и q, осуществить их перебор и подобрать закрытый ключ для зашифрованного документа.**

При решении данной задачи предполагаем, что нам известен открытый ключ <n,e> и примерные границы в которых находятся числа p и q. Мы знаем число n, множителями которого являются p и q. Также нам известно, что p и q – простые числа примерно одного порядка, т.е. находятся относительно недалеко друг от друга.

В программе мы использовали условие if n%i == 0 and a <=i <=b

Чтобы отсеять неподходящие делители. Далее с помощью библиотеки itertools и команды combinations мы находим все сочетания найденных множителей. Далее для каждой пары находим соответствующее **d** и расшифровываем файл. Результаты расшифровки по всем парам p и q записываются в файл result.txt

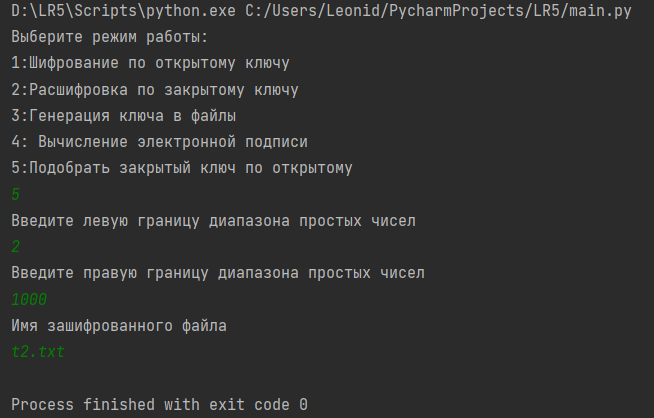


Рисунок – Работа в режиме подбора закрытого ключа

Поскольку для шифрования мы использовали только первые 100 случайны чисел, они не превышали 1000, поэтому вариант расшифровки получился единственный:

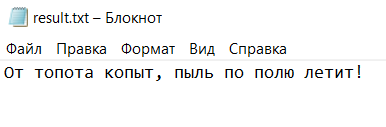


Рисунок – Расшифровка по подобранным значениям

**7.Сделать выводы о свойствах, которыми должны обладать криптографические алгоритмы вычисления контрольных сумм и ключи для алгоритма RSA.**

При вычислении контрольной суммы имеет смысл искать сумму по модулю 2 не для всех байтов подряд, а комбинируя их по группам. Например, есть последовательность байтов: 20\30\40\15\25\35. Если следовать логике, по которой мы делали 4-5 пункты данной лабораторной работы, checksum = 20^30^40^15^25^35 = 23

Если же комбинировать хотя бы по три байта, то контрольная сумма будет зависеть от порядка байтов. Изменить файл без изменения контрольной суммы в таком случае так просто не выйдет, поскольку (20+30+40) ^ (15+25+35) = 17

Ключи RSA должны быть достаточно большими простыми числами. В таком случае подбор значений p и q при известном открытом ключе становится затруднительной задачей, поскольку требует больших вычислительных мощностей. В рассмотренном примере мы брали p и q относительно небольшими числами и поэтому подбор не занял много времени.

Всё дело в том, что компьютеру необходимо проверить числа от 2 до 1000, тогда как в настоящих ключах RSA могут использоваться числа для записи которых нужно более 100 знаков.

Код программы для выполнения заданий лабораторной работы:

import sympy # Библиотека для работы с математикой  
import random  
from itertools import \*  
  
# Выбираем режим работы программы  
mode = int(input(  
 'Выберите режим работы:\n1:Шифрование по открытому ключу\n2:Расшифровка по закрытому ключу\n3:Генерация ключа в файлы\n4: Вычисление электронной подписи\n5:Подобрать закрытый ключ по открытому\n'))  
match mode:  
 case 3:  
 # Генерация ключей  
 f\_open\_key = open('open\_key.txt', 'w')  
 f\_close\_key = open('close\_key.txt', 'w')  
 primes = [] # В этот массив запишем первые 100 простых чисел, начиная с 17  
 i = 0  
 k = 17  
 while i != 100: # пока не набрано 100 чисел  
 if sympy.isprime(k): # Проверяем простое ли число  
 primes.append(k) # Если простое то добавляем в массив  
 i += 1  
 k += 1  
 else: # Иначе проверяем следующее число  
 k += 1  
 num = random.randint(0, len(primes) - 2) # Из получившихся чисел выбираем случайное и присваиваем p  
 p = primes[num]  
 q = primes[num + 1] # Присваиваем q значение следующего простого числа после случайно выбранного  
 n = p \* q # вычисляем n и фи  
 phi = (p - 1) \* (q - 1)  
 delit = [] # Массив делителей числа фи  
 for i in range(2, phi): # Перебираем множители  
 if sympy.gcd(i, phi) == 1 and sympy.isprime(i): # Если НОД фи и множителя равен 1, добавляем в массив  
 delit.append(i)  
 e = delit[  
 random.randint(0, len(delit) - 1)] # Выбираем e случайным взаимнопростым числом из сформированного списка  
 d = 0  
 u = 1  
 while d == 0: # Ищем d из условия d \* e - u \* phi = 1  
 if (phi \* u + 1) % e == 0:  
 d = int((phi \* u + 1) / e)  
 else:  
 u += 1  
 f\_open\_key.write(bin(n) + bin(e)) # записываем в файл открытый ключ в бинарном формате  
 f\_close\_key.write(bin(n) + bin(d)) # записываем закрытый ключ в бинарном формате в файл  
 f\_open\_key.close()  
 f\_close\_key.close()  
  
 ## Шифрование  
 case 1:  
 f\_text = open(input('Имя файла,который нужно зашифровать\n'), 'rb')  
 fko = open('open\_key.txt', 'r')  
 f\_cr = open(input('Имя зашифрованного файла\n'), 'w')  
 ne = fko.read().split('0b')[1:] # считываем открытый ключ из файла  
 n1, e1 = int(ne[0], 2), int(ne[1], 2) # присваиваем n и е целочисленные значения  
 bits = ''  
 text = f\_text.read() # Читаем байты открытого текста из файла  
 crypt\_int = []  
 for i in text:  
 crypt\_int.append(i \*\* e1 % n1) # Шифруем полученные числа с помощью открытого ключа  
 out = ''  
 for j in crypt\_int: # Для хранения зашифрованных данных переводим числа в бинарный формат и записываем в файл  
 out += bin(j)  
 f\_cr.write(out)  
 f\_text.close()  
 fko.close()  
 f\_cr.close()  
  
 case 2:  
 ## Расшифровка  
 f1 = open(input('Что расшифровать?\n'), 'r')  
 f2 = open(input('Куда записать результат?\n'), 'wb')  
 fkey = open('close\_key.txt', 'r')  
 nd2 = fkey.read().split('0b')[1:] # Считываем закрытый ключ  
 n2, d2 = int(nd2[0], 2), int(nd2[1], 2) # Присваиваем n и d  
 s1 = f1.read().split('0b')[1:] # Считаем содержимое зашифрованного файла  
 cr\_int = []  
 for i in s1: # Преобразуем бинарную запись в числовую  
 cr\_int.append(int(i, 2))  
 decr\_int = []  
 for i in cr\_int:  
 decr\_int.append((i \*\* d2) % n2) # Расшифровываем числа с помощью d и n - закрытого ключа  
 for i in decr\_int: # битовую запись разбиваем по 8 бит на число, обозначающее байт открытого текста и записываем в файл  
 f2.write(bytes([i]))  
 f1.close()  
 f2.close()  
 fkey.close()  
  
 case 4:  
 # Вычисление контрольной суммы  
 file = open(input('Для чего считать контрольную сумму?\n'), 'rb')  
 fileout = open(input('Куда сохранить результат?\n'), 'w')  
 keyfile = open('close\_key.txt', 'r')  
 nd3 = keyfile.read().split('0b')[1:] # Считываем закрытый ключ  
 n3, d3 = int(nd3[0], 2), int(nd3[1], 2)  
 numbers = file.read() # Считываем байты файла  
 checksum = 0  
 for i in numbers: # Вычисляем контрольную сумму, используя операцию xor для каждого байта последовательно  
 checksum = checksum ^ i  
 C = (checksum \*\* d3) % n3 # Зашифровываем закрытым ключом значение контрольной суммы  
 fileout.write(str(C))  
 file.close()  
 fileout.close()  
 keyfile.close()  
 case 5:  
 a = int(input('Введите левую границу диапазона простых чисел\n'))  
 b = int(input('Введите правую границу диапазона простых чисел\n'))  
 tfile = open(input('Имя зашифрованного файла\n'), 'r')  
 fil = open('open\_key.txt', 'r')  
 result = open('result.txt', 'wb')  
 ne = fil.read().split('0b')[1:] # считываем открытый ключ из файла  
 n, e = int(ne[0], 2), int(ne[1], 2) # Получаем открытый ключ в целых числах  
 deliteli, cr\_int = [], []  
 sh = tfile.read().split('0b')[1:] # Получаем список двоичных представлений зашифрованных чисел  
 for i in sh:  
 cr\_int.append(int(i, 2)) # Переводим двоичные числа в int  
 for i in range(a, n): # Ищем делители числа n, попадающие в границы указанного диапазона  
 if n % i == 0 and i <= b:  
 deliteli.append(i)  
 pairs = []  
 for i in combinations(deliteli[:],2): # Находим список сочетаний полученных множителей  
 pairs.append(list(i))  
 for i in pairs: # Для каждой пары p и q вычисляем d и phi  
 phi = (i[0] - 1) \* (i[1] - 1)  
 d = 0  
 u = 1  
 while d == 0: # Ищем d из условия d \* e - u \* phi = 1  
 if (phi \* u + 1) % e == 0:  
 d = int((phi \* u + 1) / e)  
 else:  
 u += 1  
 for o in cr\_int: # для найденной d и известных e и n получаем расшифровку и записываем в файл  
 result.write(bytes([(o \*\* d) % n]))  
 result.write(  
 b'\r\n') # После записи байтов исходного текста вводим символ новой строки, чтобы отделить варианты расшифровки  
 fil.close()  
 result.close()  
 case \_: # Некорректный ввод режима работы программы завершает работу с кодом ошибки  
 exit(1)