|  |  |
| --- | --- |
| **Российский университет транспорта (МИИТ)**  **Институт транспортной техники и систем управления**  **Кафедра «Управление и защита информации»** | |
| **Отчет**  **по курсовому проекту**  **по теме «Исследование и реализация методов ассиметричной криптографии»**  **по дисциплине «Криптографические методы защиты информации»** | |
|  | Выполнил:  Студент группы ТКИ-342  Дроздов А.Д.  Проверил:  Доцент кафедры УиЗи, к.т.н., с.н.с.  Михалевич И.Ф. |
| Москва 2023 | |

Оглавление

[Задание 4](#_Toc136384549)

[Исходные данные 4](#_Toc136384550)

[1. Вычисление простого числа 5](#_Toc136384551)

[1.1. Теоретическая часть 5](#_Toc136384552)

[1.1.1. Решето Эратосфена 5](#_Toc136384553)

[1.1.2. Решето Аткина 6](#_Toc136384554)

[1.1.3. Числа Марсенна и тест Люка-Лемера 7](#_Toc136384555)

[1.2. Реализация метода проверки числа на простоту 8](#_Toc136384556)

[2. Проверка чисел на взаимную простоту 9](#_Toc136384557)

[2.1. Теоретическая часть 9](#_Toc136384558)

[2.1.1. Взаимно простые числа 9](#_Toc136384559)

[2.1.2. Алгоритм Евклида 10](#_Toc136384560)

[2.1.3. Разложение на множители 11](#_Toc136384561)

[2.2. Реализация метода проверки чисел на взаимную простоту 12](#_Toc136384562)

[3. Описание метода RSA 13](#_Toc136384563)

[4. Реализация метода RSA 13](#_Toc136384564)

[4.1. Шифрование и расшифрование RSA 14](#_Toc136384565)

[4.2. Формирование и проверка ЭЦП 15](#_Toc136384566)

[4.3. Реализация метода в Excel 15](#_Toc136384567)

[4.2. Реализация метода с помощью программы 16](#_Toc136384568)

[4.2.1. Код программы – основная часть 17](#_Toc136384569)

[4.2.2. Код программы – подпрограмма поиска простых чисел 21](#_Toc136384570)

[4.2.3. Код программы – подпрограмма проверки чисел на взаимную простоту 22](#_Toc136384571)

[4.2.4. Результат работы программы 23](#_Toc136384572)

[5. Область применения метода RSA 24](#_Toc136384573)

[6. Заключение 24](#_Toc136384574)

# Задание

1. Вычислить простое число *p* (привести описание трех или более методов проверки простоты, реализовать один или более, проверить число на простоту)

2. Проверить числа *l* и *(b + f)* на взаимную простоту (привести описание методов проверки взаимной простоты чисел, реализовать один или более, проверить, являются ли числа *l* и *(b + f)* взаимно простыми)

3. Описать метод (дать общую характеристику исследуемого метода)

4. Реализовать метод (в программе Excel и с помощью программы)

5. Описать области применения метода (привести примеры)

6. Оформить отчет

# Исходные данные

На рис.1 представлены исходные данные, где *b* – это восьмиразрядное число, отражающее день рождения (в формате день, месяц, год полностью);   
*f* – это восьмиразрядное число, которое высчитывается как сумма порядкового номера и 10, а далее приписывается шесть нулей; *p* – это простое меньшее ближайшее число к числу, равному сумме *b* и *f*; *l* – это шестиразрядное число, равное числу *b* с исключением первого и третьего разряда.

|  |
| --- |
|  |
| 1. – Исходные данные |

В данной работе будет исследован ассиметричный метод   
шифрования RSA.

# 1. Вычисление простого числа

## 1.1. Теоретическая часть

Натуральное число называется простым, если оно имеет только два делителя: единицу и само себя. Задача поиска простых чисел не дает покоя математикам уже очень давно. Долгое время прямого практического применения эта проблема не имела, но все изменилось с появлением криптографии с открытым ключом. Далее рассматривается несколько способов поиска простых чисел.

### 1.1.1. Решето Эратосфена

[Решето Эратосфена](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D1%88%D0%B5%D1%82%D0%BE_%D0%AD%D1%80%D0%B0%D1%82%D0%BE%D1%81%D1%84%D0%B5%D0%BD%D0%B0) — алгоритм, предложенный древнегреческим математиком Эратосфеном, который позволяет найти все простые числа меньше заданного целого числа *n*.

Для нахождения простых чисел в алгоритме Эратосфена последовательно выполняются несколько шагов. Рассмотрим их.

Первым шагом задаётся целое числа *n*, до которого нужно найти простые числа. Далее выписываются подряд все числа от 2 до *n*.

Второй шаг – объявление переменной *p*, которая изначально будет равна двум – это первое простое число в заданном диапазоне чисел.

Третий шаг – пройтись по всем числам и вычеркнуть такие числа, которые кратны числу *p* (кроме *p*).

Четвертым шагом необходимо присвоить переменной *p* первое оставшееся число в списке, которое больше ранее заданного значения *p*.

Последний шаг – это выполнение третьего и четвертого шага, пока это возможно. В итоге будет получен список простых чисел до заданного целого числа *n*.

Такой алгоритм можно оптимизировать – так как один из делителей составного числа *n* обязательно , алгоритм можно останавливать, после вычеркивания чисел делящихся на .

Сложность алгоритма составляет , при этом, для хранения информации о том, какие числа были вычеркнуты требуется  памяти.

### 1.1.2. Решето Аткина

Более совершенный алгоритм отсеивания составных чисел был предложен Аткином и Берштайном и получил название [Решето Аткина](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D1%88%D0%B5%D1%82%D0%BE_%D0%90%D1%82%D0%BA%D0%B8%D0%BD%D0%B0). Этот способ основан на следующих трех свойствах простых чисел.

Свойство 1 – если *n* – положительное число, не кратное квадрату простого числа и такое, что . То *n* — простое, тогда и только тогда, когда число корней уравнения нечетно.

Свойство 2 – если *n* – положительное число, не кратное квадрату простого числа и такое, что . То *n* — простое, тогда и только тогда, когда число корней уравнения  нечетно.

Свойство 3 – если *n* – положительное число, не кратное квадрату простого числа и такое, что . То *n* — простое, тогда и только тогда, когда число корней уравнения  нечетно.

На начальном этапе алгоритма решето Аткина представляет собой массив *A* размером *n*, заполненный нулями. Для определения простых чисел перебираются все . Для каждой такой пары вычисляется , ,  и значение элементов массива , , увеличивается на единицу. В конце работы алгоритма индексы всех элементов массива, которые имеют нечетные значения либо простые числа, либо квадраты простого числа. На последнем шаге алгоритма производится вычеркивание квадратов оставшихся в наборе чисел.

Из описания алгоритма следует, что вычислительная сложность решета Аткина и потребление памяти составляют . При использовании «wheel factorization» и сегментирования оценка сложности алгоритма снижается до , а потребление памяти до .

### 1.1.3. Числа Марсенна и тест Люка-Лемера

Конечно при таких показателях сложности, даже оптимизированное решето Аткина невозможно использовать для поиска по-настоящему больших простых чисел. К счастью, существуют быстрые тесты, позволяющие проверить является ли заданное число простым. В отличие от алгоритмов решета, такие тесты не предназначены для поиска всех простых чисел, они лишь способны сказать с некоторой вероятностью, является ли определенное число простым.

Один из таких методов проверки — [тест Люка-Лемера](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82_%D0%9B%D1%8E%D0%BA%D0%B0_%E2%80%94_%D0%9B%D0%B5%D0%BC%D0%B5%D1%80%D0%B0). Это детерминированный и безусловный тест простоты. Это означает, что прохождение теста гарантирует простоту числа. К сожалению, тест предназначен только для чисел особого вида , где *p* — натуральное число. Такие числа называются числами Мерсенна.

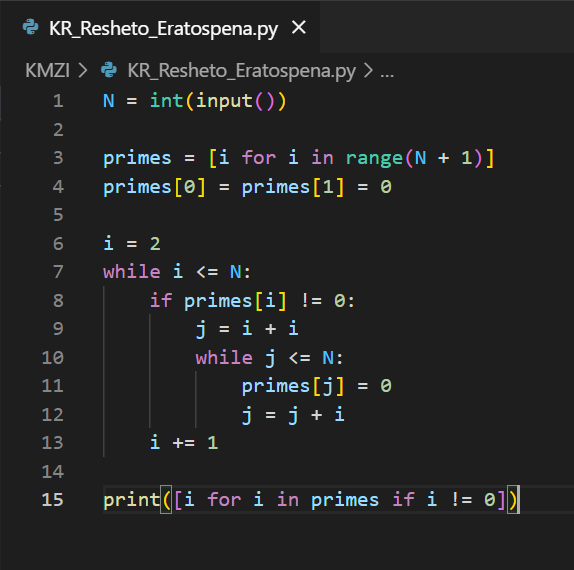
Тест Люка-Лемера утверждает, что число Мерсенна простое тогда и только тогда, когда *p* — простое и делит нацело -й член последовательности  задаваемой рекуррентно:  для .

Для числа длиной *p* бит вычислительная сложность алгоритма составляет .

## 1.2. Реализация метода проверки числа на простоту

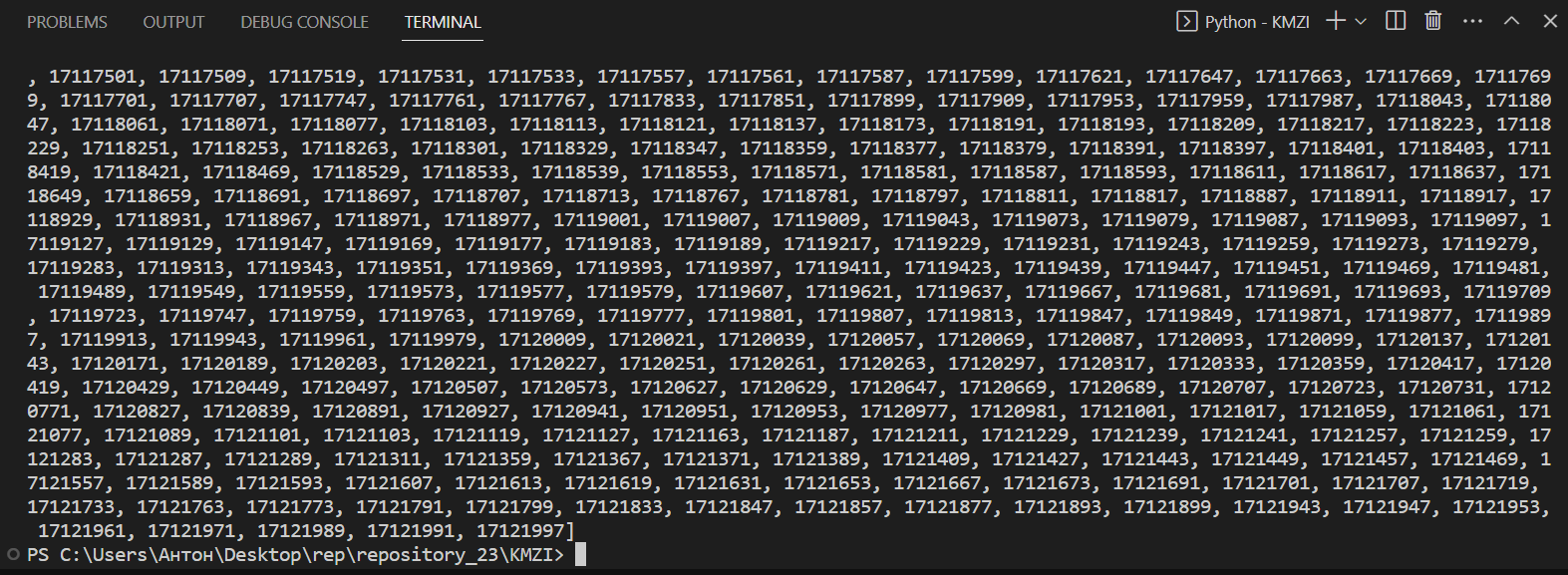
Рассмотрим реализацию одного из алгоритмов поиска простых чисел – Решето Эратосфена.

Код программы на языке Python изображен на рисунке 2.



1. – Код программы (Решето Эратосфена)

Далее, на рисунке 3 представлен результат работы программы по нахождению ближайшего меньшего простого числа к числу, равному сумме *d + r* – заданы в пункте "Исходные данные".



1. – Результат работы программы (Решето Эратосфена)

# 2. Проверка чисел на взаимную простоту

## 2.1. Теоретическая часть

### 2.1.1. Взаимно простые числа

Два целых числа a и b называются взаимно простыми, если их наибольший общий делитель равен единице — то есть НОД (a, b) = 1.

Из определения взаимно простых чисел можно сделать вывод, что у двух взаимно простых чисел может быть только один положительный общий делитель, который равен единице. А всего общих делителей у двух взаимно простых чисел два — это 1 и -1.

Приведем примеры взаимно простых чисел.

Числа 13 и 16 взаимно простые потому, что их положительный общий делитель — единица, что подтверждает взаимную простоту чисел 13 и 16. Заметим, что два простых числа всегда являются взаимно простыми. Однако, два числа не обязательно должны быть простыми, чтобы быть взаимно простыми.

Приведем пример.

Два составных числа 8 и -9 являются взаимно простыми. Сначала найдем НОД этих чисел.

Делители 8: ±1, ±2, ±4, ±8.

Делители -9: ±1, ±3, ±9.

Из этого следует, НОД (8, -9) = 1, поэтому, по определению 8 и -9 — два взаимно простых числа.

Так же это работает, когда у нас не два числа, а больше.

Взаимно простыми целые числа будут тогда, когда они имеют наибольший общий делитель, равный 1.

Иными словами, если у нас есть набор некоторых чисел с наибольшим положительным делителем, большим 11, то все эти числа не являются по отношению друг к другу взаимно обратными.

Возьмем несколько примеров. Так, целые числа −99, 17 и −27 – взаимно простые. Любое количество простых чисел будет взаимно простым по отношению ко всем членам совокупности, как, например, в последовательности 2, 3, 11, 19, 151, 293 и 667. А вот числа 12, −9, 900 и −72 взаимно простыми не будут, потому что кроме единицы у них будет еще один положительный делитель, равный 33. То же самое относится к числам 17, 85 и 187: кроме единицы, их все можно разделить на 17.

Обычно взаимная простота чисел не является очевидной с первого взгляда, этот факт нуждается в доказательстве. Чтобы выяснить, будут ли некоторые числа взаимно простыми, нужно найти их наибольший общий делитель и сделать вывод на основании его сравнения с единицей.

Рассмотрим два основных метода нахождения НОД двумя основными способами: с использованием алгоритма Евклида и путем разложения на простые множители.

### 2.1.2. Алгоритм Евклида

Алгоритм Евклида помогает найти НОД через последовательное деление.

Алгоритм Евклида заключается в следующем: если большее из двух чисел делится на меньшее — наименьшее число и будет их наибольшим общим делителем. Использовать метод Евклида можно легко по формуле нахождения наибольшего общего делителя.

Формула НОД:

НОД (a, b) = НОД (b, с), где с — остаток от деления a на b.

Для нахождения наибольшего общего делителя двух чисел нужно соблюдать такой порядок действий:

1. Большее число поделить на меньшее.
2. Меньшее число поделить на остаток, который получается после деления.
3. Первый остаток поделить на второй остаток.
4. Второй остаток поделить на третий и т. д.

Деление продолжается до тех пор, пока в остатке не получится нуль. Последний делитель и есть наибольший общий делитель.

Пример. Найти наибольший общий делитель чисел 140 и 96:

140 : 96 = 1 (остаток 44)

96 : 44 = 2 (остаток 8)

44 : 8 = 5 (остаток 4)

8 : 4 = 2

Последний делитель равен 4 — это значит: НОД (140, 96) = 4.

Ответ: НОД (140, 96) = 4

### 2.1.3. Разложение на множители

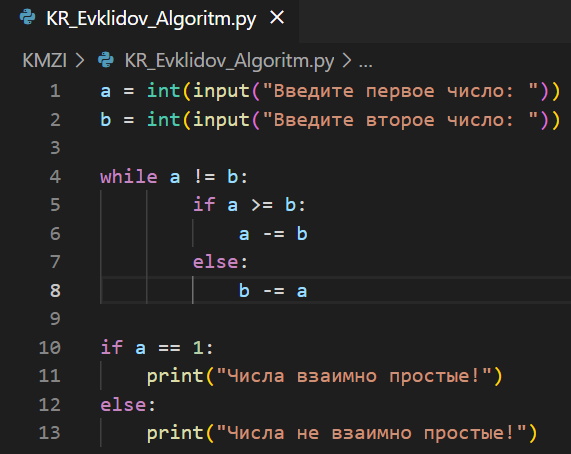
Для того, чтобы найти наибольший общий делитель двух чисел методом разложения на множители, необходимо перемножить все простые множители, которые получаются при разложении этих двух чисел и являются для них общими.

Пример.

Если мы разложим числа 220 и 600 на простые множители, то получим два произведения: 220=2⋅2⋅5⋅11 и 600=2⋅2⋅2⋅3⋅5⋅5. Общими в этих двух произведениях будут множители 2,2 и 5. Это значит, что НОД(220, 600)=2⋅2⋅5=20.

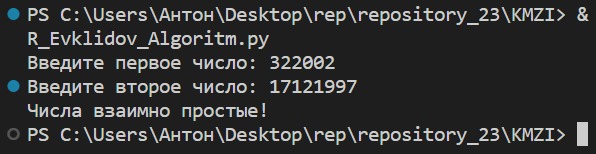
## 2.2. Реализация метода проверки чисел на взаимную простоту

Программа проверки чисел на взаимную простоту с помощью алгоритма Евклида – рисунок 4.



1. – Код программы (Алгоритм Евклида)

На рисунке 5 – представление работы программы – проверка исходных данных на взаимную простоту.



1. – Результат программы (Алгоритм Евклида)

# 3. Описание метода RSA

Сегодня метод криптографической защиты данных с открытым ключом RSA используется часто. Своё название получил по начальным буквам фамилии своих изобретателей – Rivest, Shamir, Adleman. На основе метода RSA разработаны алгоритмы шифрования, успешно применяемые для защиты информации. Он обладает высокой криптостойкостью и может быть реализован при использовании относительно несложных программных и аппаратных средств.

Функционирование криптосистемы на основе метода RSA осуществляется с помощью открытого и секретного ключа.

# 4. Реализация метода RSA

Метод RSA используется для реализации шифрования/расшифрования и проверки подлинности – то есть, использование электронной цифровой подписи (ЭЦП).

Для реализации вышеуказанных возможностей RSA необходимо подготовить следующие параметры:

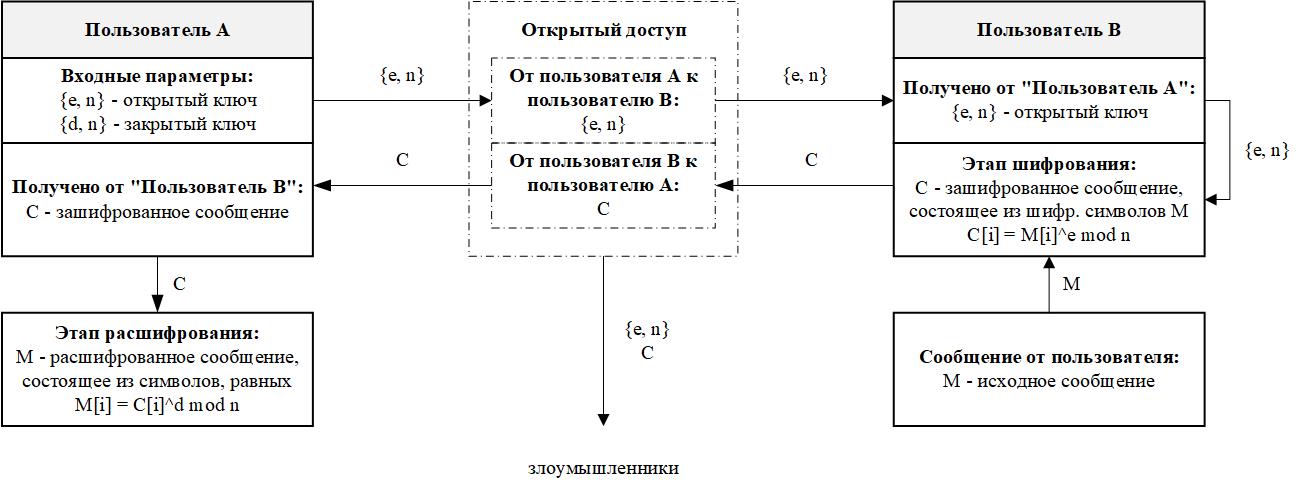
1. Два простых числа и .
2. Число , равное произведению и .
3. Функция Эйлера, на вход которая принимает значение числа – **ф(n) = (p-1) × (q-1)**.
4. Число , выбор которого основывается на критериях: – простое,  **ф(n)**, взаимно простое с **ф(n)**.
5. Такое число , чтобы  **% ф(n) = 1** – для удобства лучше взять число отличное от .

В конечном результате имеем два ключа – открытый ключ **{e, n}** и секретный ключ **{d, n}**.

Далее рассмотрим возможности RSA с помощью схем.

## 4.1. Шифрование и расшифрование RSA

На рисунке 6 представлена схема шифрования и расшифрования сообщения.



1. – Шифрование и расшифрование методом RSA

Алгоритм шифрования сообщения методом RSA следующий: "Пользователь В" получает от "Пользователь А" открытый ключ (e, n). Символы входного сообщения *M*, применив полученный ранее открытый ключ, шифруются:

|  |  |
| --- | --- |
|  |  |

В результате зашифрованное сообщение *С* отправляется обратно к "Пользователь А".

Следующий этап – это посимвольное расшифрование полученного сообщения. Этим занимается "Пользователь А", использующий секретный ключ:

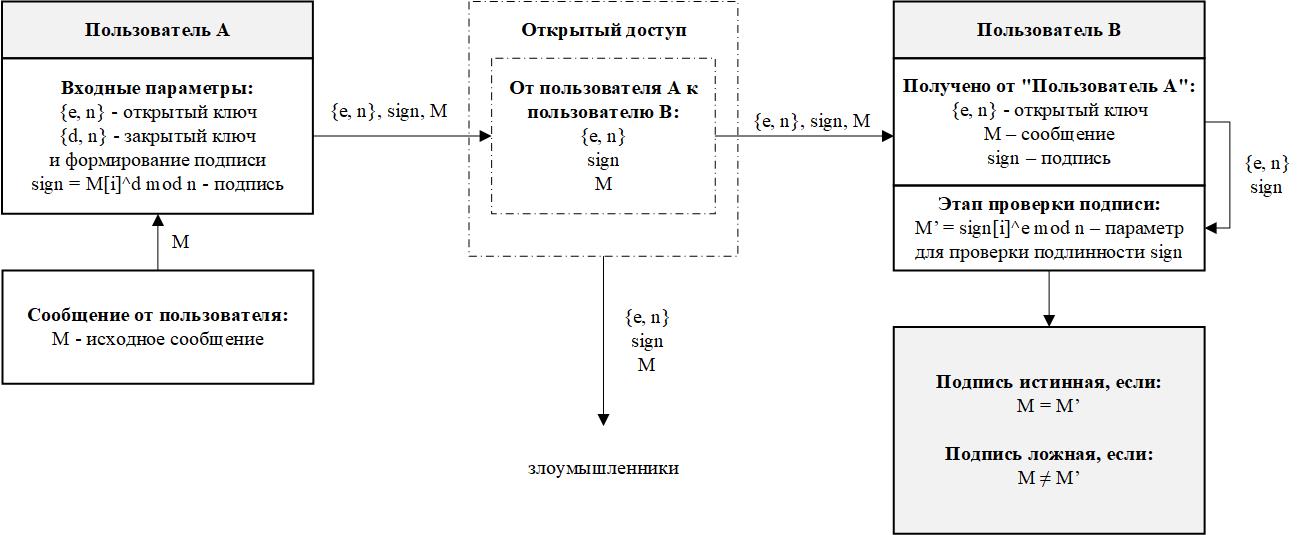
|  |  |
| --- | --- |
|  |  |

В результате "Пользователь А" расшифровал полученное сообщение от "Пользователь В".

Важно отметить, что "Пользователь А" никогда и никому не отправляет свой секретный ключ.

## 4.2. Формирование и проверка ЭЦП

На рисунке 7 представлена схема формирования и проверка подлинности ЭЦП.



1. – Формирование и проверка подлинности ЭЦП методом RSA

Алгоритм формирования ЭЦП методом RSA следующий: "Пользователь А" отправляет "Пользователь В" открытый ключ (e, n), сообщение M и подпись sign, которая была получена следующим образом (d – это составляющая секретного ключа user\_1):

|  |  |
| --- | --- |
|  |  |

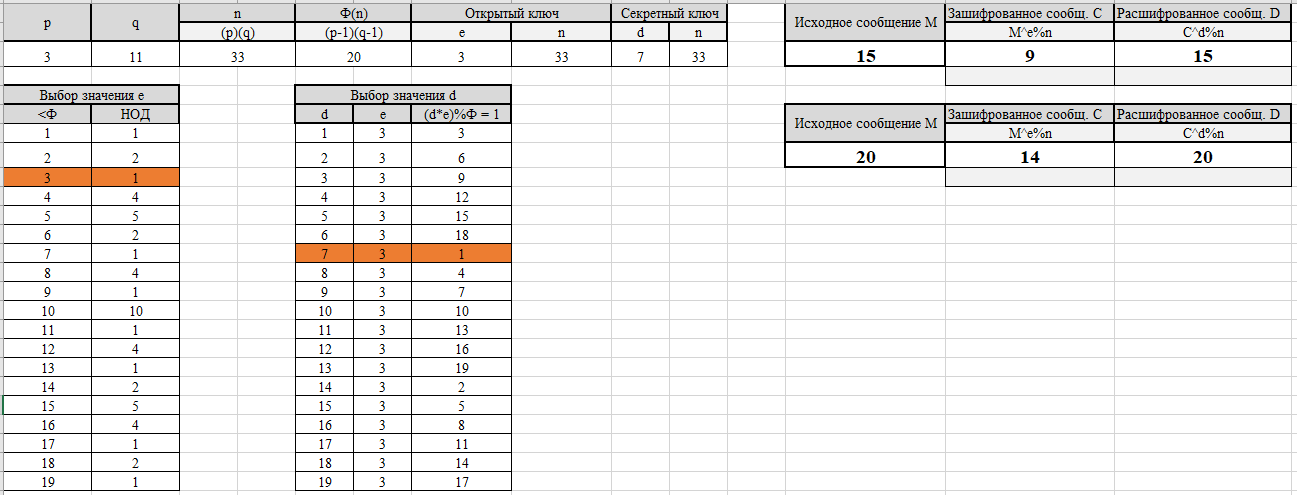
Далее осуществляется проверка подписи. Для этого получателю ("Пользователь В") необходимо полученное значение sign возвести в степень e из открытого ключа по модулю n и сравнить со значением полученного M:

|  |  |
| --- | --- |
|  |  |

В случае, если - подпись прошла проверку, то есть она истинная. Иначе, если - подпись ложная.

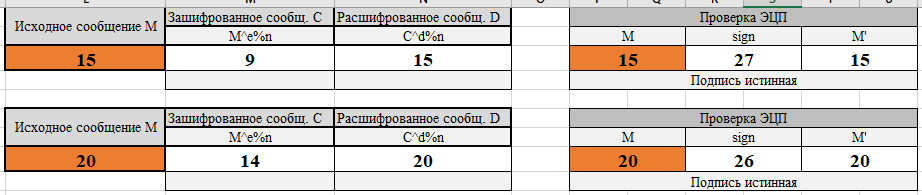
## 4.3. Реализация метода в Excel

Один из вариантов реализации метода – это реализация в Excel. На рисунке 8 представлено 2 примера шифрования и расшифрования исходных сообщений (первое сообщение – 15, второе – 20) с помощью метода RSA.



1. – Реализация метода RSA в Excel

Также в Excel было реализовано формирование ЭЦП и проверка её подлинности – рисунок 9 (в данном примере M – является числом, а значит формирование подписи происходит не по символу).



1. **–** Формирование и проверка ЭЦП

## 4.2. Реализация метода с помощью программы

Реализация метода шифрования RSA выполнялась на языке программирования Python.

Структура реализации метода RSA следующая:

1. Подпрограмма проверки чисел на простоту (Решето Эратосфена);
2. Подпрограмма проверки чисел на взаимную простоту   
   (алгоритм Евклида);
3. Основная программа, состоящая из подпрограмм (пункт 1, 2), функций для шифрования и расшифрования сообщения, их вывода, а также сделана реализация ЭЦП.

### 4.2.1. Код программы – основная часть

import KR\_Resheto\_Eratospena, KR\_Algoritm\_Evklida

p = int(input('\nВведите число p: '))

q = int(input('Введите число q: '))

message = input('Введите сообщение, которое нужно зашифровать: ')

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ПАРАМЕТРЫ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

n = p \* q

f\_n = (p-1) \* (q-1)

e = 0

d = 0

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ПАРАМЕТРЫ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

def open\_key(func\_eiler): # ГЕНЕРАЦИЯ ОТКРЫТОГО КЛЮЧА

    primes = KR\_Resheto\_Eratospena.resheto\_eratosphena(func\_eiler)

    for i in range(len(primes)):

        if KR\_Algoritm\_Evklida.algoritm\_evklida(primes[i],f\_n) != 1:

            primes[i] = 0

    primes = [i for i in primes if i != 0]

    return primes[0]

e = open\_key(f\_n)

def secret\_key(func\_eiler, e\_from\_open\_key): # ГЕНЕРАЦИЯ СЕКРЕТНОГО КЛЮЧА

    d\_list = [] # Список для выбора значения d

    d\_ok = []

    for i in range(func\_eiler):

        d\_list.append(i+1)

    for d in d\_list:

        if d \* e\_from\_open\_key % func\_eiler == 1:

            d\_ok.append(d)

    return d\_ok[0]

d = secret\_key(f\_n, e)

def encryption(position\_inp\_message): # ЗАШИФРОВАНИЕ

    position\_encr\_message = []

    for position in position\_inp\_message:

        position\_encr\_message.append(position \*\* e % n)

    return position\_encr\_message

def decription(position\_enc\_message): # РАСШИФРОВАНИЕ

    position\_decr\_message = []

    for position in position\_enc\_message:

        position\_decr\_message.append(position \*\* d % n)

    return position\_decr\_message

def position\_characters(msg): # ПОЗИЦИЯ СИМВОЛА В UNICODE

    position = []

    for i in msg:

        position.append(ord(i))

    return position

def output\_message(msg\_pos: list): # ВЫВОД

    out\_msg = ''

    for i in msg\_pos:

        out\_msg += chr(i)

    print(out\_msg)

def bin\_character(out\_position): # ДВОИЧНОЕ ПРЕДСТАВЛЕНИЕ

    out\_bin\_position = []

    for i in out\_position:

        out\_bin\_position.append(bin(i))

    print(out\_bin\_position)

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ВЫВОД\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

print('\nn:', n, '\ne:', e, '\nd:', d)

print('\n\tИсходное сообщение:')

output\_message(position\_characters(message))

print(position\_characters(message))

print('Двоичный вид:')

bin\_character(position\_characters(message))

print('\n\tЗашифрованное сообщение:')

output\_message(encryption(position\_characters(message)))

print(encryption(position\_characters(message)))

print('Двоичный вид:')

bin\_character(encryption(position\_characters(message)))

print('\n\tРасшифрованное сообщение:')

output\_message(decription(encryption(position\_characters(message))))

print(decription(encryption(position\_characters(message))))

print('Двоичный вид:')

bin\_character(decription(encryption(position\_characters(message))))

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ЭЦП\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

def create\_sign(inp\_message): # СОЗДАНИЕ ПОДПИСИ

    position = position\_characters(inp\_message)

    sign = []

    for i in position:

        sign.append(i \*\* d % n)

    return sign

def check\_sign(sign): # ЭТАП ПОДГОТОВКИ ПАРАМЕТРА M' ДЛЯ ПРОВЕРКА ПОДЛИННОСТИ

    check\_signature = []

    for i in sign:

        check\_signature.append(i \*\* e % n)

    return check\_signature

def chech\_final(inp\_message, sign): # ПРОВЕРКА ПОДЛИННОСТИ

    if position\_characters(inp\_message) == sign:

        print('Подпись истинная')

    else:

        print('Подпись ложная')

print('\n\tСоздание ЭЦП')

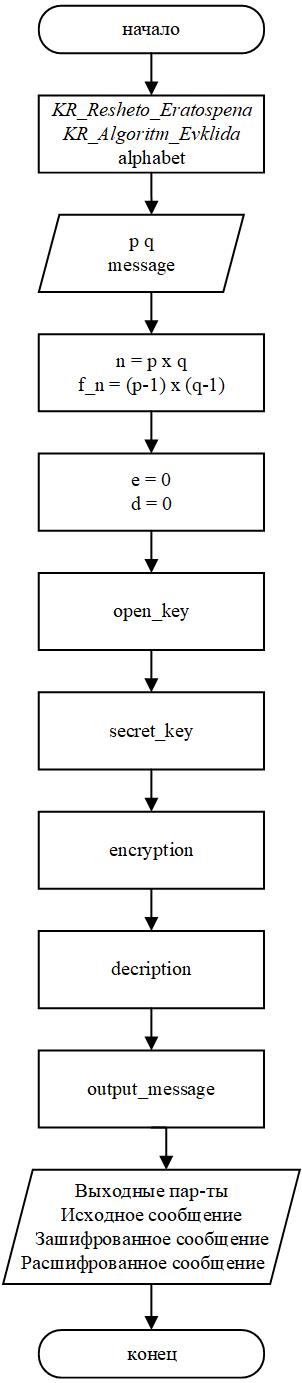
print('Сообщение M:', message, position\_characters(message))

print('sign:', create\_sign(message))

print("M':", check\_sign(create\_sign(message)))

chech\_final(message, check\_sign(create\_sign(message)))

На рисунке 10 представлена блок схема программы (4.2.1.).



1. – Блок схема основной программы

### 4.2.2. Код программы – подпрограмма поиска простых чисел

def resheto\_eratosphena(N):

    primes = [i for i in range(N + 1)]

    primes[0] = primes[1] = 0

    i = 2

    while i <= N:

        if primes[i] != 0:

            j = i + i

            while j <= N:

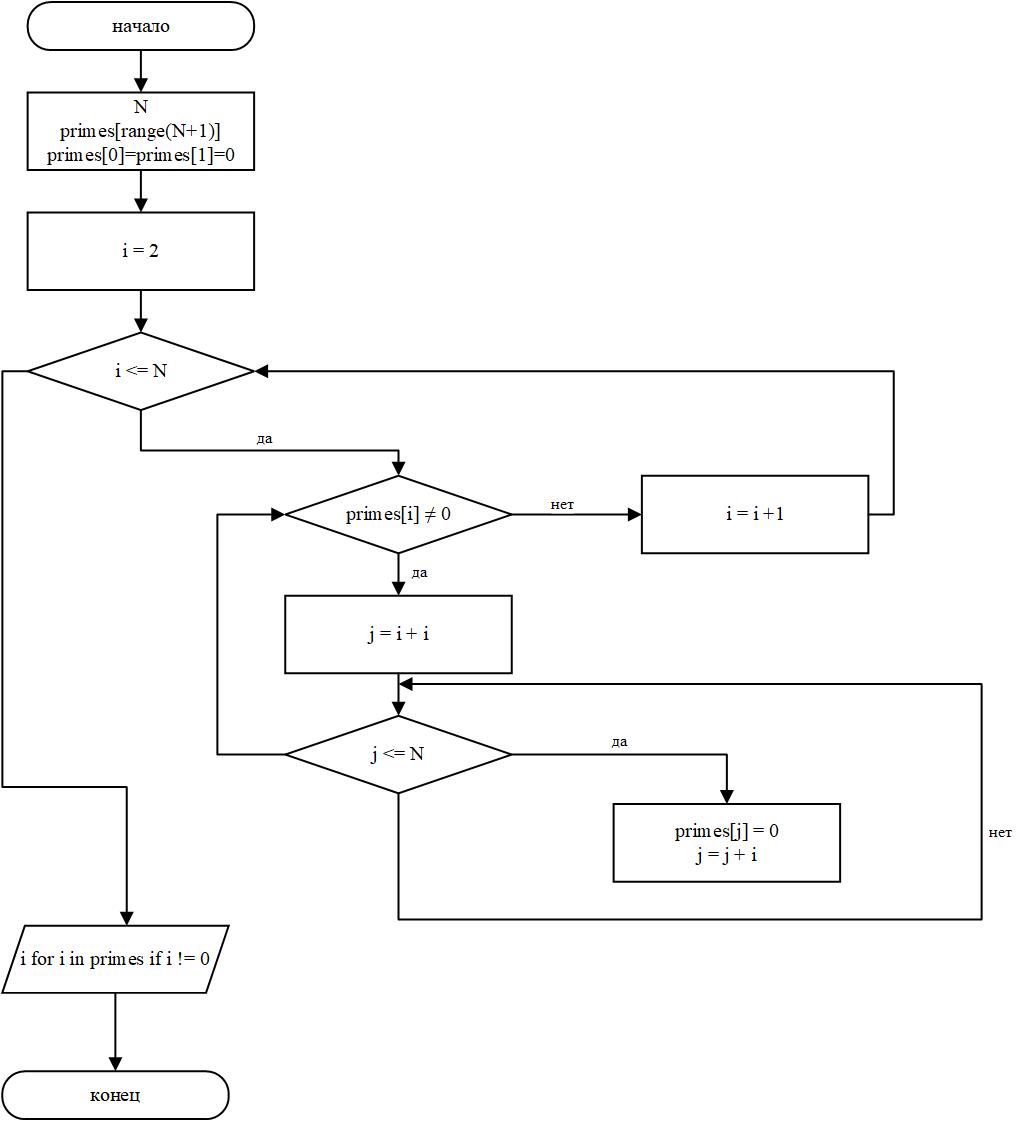
                primes[j] = 0

                j = j + i

        i += 1

    return [i for i in primes if i != 0]

На рисунке 11 – блок схема для алгоритма поиска простых числе (4.2.2).



1. – Блок схема (Решето Эратосфена)

### 4.2.3. Код программы – подпрограмма проверки чисел на взаимную простоту

def algoritm\_evklida(num\_1, num\_2):

    while num\_1 != num\_2:

            if num\_1 >= num\_2:

                num\_1 -= num\_2

            else:

                num\_2 -= num\_1

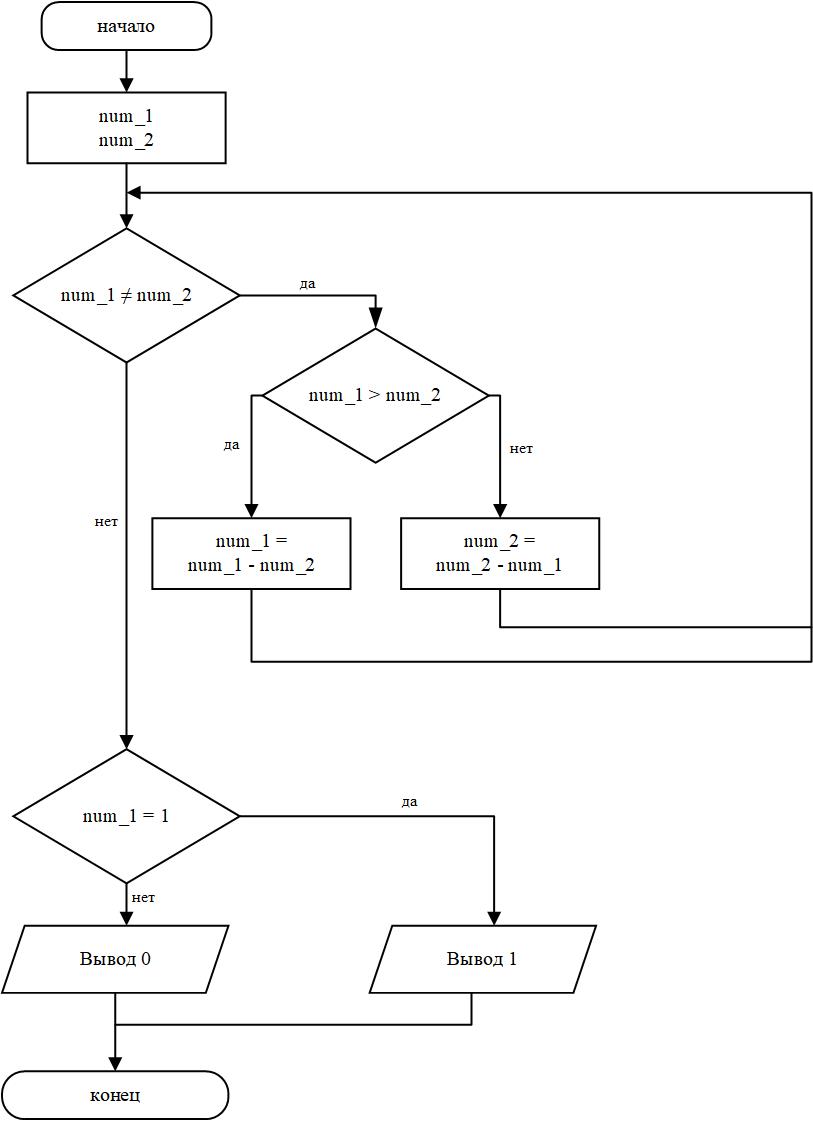
    if num\_1 == 1:

        return 1

    else:

        return 0

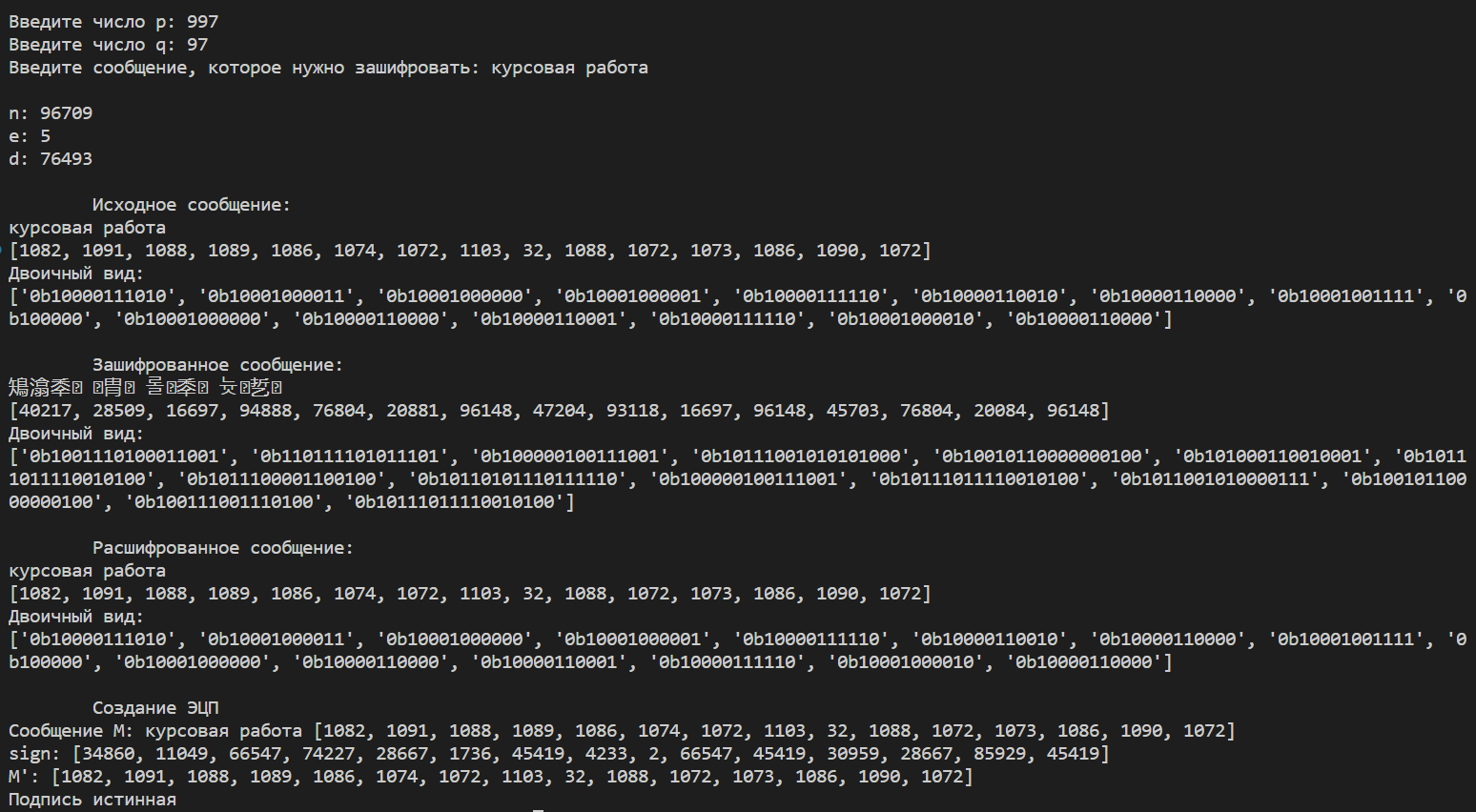
На рисунке 12 представлена блок схема Алгоритма Евклида.



1. – Блок схема (Алгоритм Евклида)

### 4.2.4. Результат работы программы

На рисунке 13 представлен результат работы программы.



1. – Результат работы программы (метод RSA)

# 5. Область применения метода RSA

Криптосистема RSA имеет применение во многих отраслях и в самых различных продуктах. В настоящее время криптосистема RSA встраивается во многие коммерческие продукты, число которых постоянно увеличивается. Также ее используют операционные системы Microsoft, Apple и другие. В аппаратном исполнении RSA алгоритм применяется в защищенных телефонах, на сетевых платах Ethernet, на смарт-картах, широко используется в криптографическом оборудовании Zaxus (Racal). Кроме того, алгоритм входит в состав всех основных протоколов для защищенных коммуникаций Internet, в том числе S/MIME, SSL и S/WAN.

Технологию шифрования RSA BSAFE используют около 500 миллионов пользователей всего мира. Так как в большинстве случаев при этом используется алгоритм RSA, то его можно считать наиболее распространенной криптосистемой общего ключа в мире и это количество имеет явную тенденцию к увеличению.

# 6. Заключение

В данной работе рассмотрен один из методов асимметричного шифрования – шифр RSA, являющийся простым для понимания и шифрования каких-либо данных. Шифр RSA используется также при создании электронно-цифровой подписи (ЭЦП).

Результатом работы является реализация данного метода шифрования и рассмотрение его особенностей. Стоит отметить недостаток данного алгоритма – это долгий процесс шифрования/расшифрования, нежели в алгоритмах симметричного шифрования. Именно поэтому RSA зачастую используется с другими алгоритмами шифрования.