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

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.

b	*f*	*p*	*l*
03122002	14000000	17121997	322002

Рисунок 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 — 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 $\leq \sqrt{n}$, алгоритм можно останавливать, после вычеркивания чисел делящихся на $\sqrt{n}$.

Сложность алгоритма составляет $O(n * \log(\log(n)))$, при этом, для хранения информации о том, какие числа были вычеркнуты требуется $O(n)$ памяти.

1.1.2. Решето Аткина

Более совершенный алгоритм отсеивания составных чисел был предложен Аткином и Берштайном и получил название Решето Аткина. Этот способ основан на следующих трех свойствах простых чисел.

Свойство 1 – если n – положительное число, не кратное квадрату простого числа и такое, что $n = 1(mod\ 4)$. То n — простое, тогда и только тогда, когда число корней уравнения $4x^2 + y^2 = n$ нечетно.

Свойство 2 – если n – положительное число, не кратное квадрату простого числа и такое, что $n = 1(mod\ 6)$. То n — простое, тогда и только тогда, когда число корней уравнения $3x^2 + y^2 = n$ нечетно.

Свойство 3 – если n – положительное число, не кратное квадрату простого числа и такое, что $n = 11(mod\ 12)$. То n — простое, тогда и только тогда, когда число корней уравнения $3x^2 - y^2 = n$ нечетно.

На начальном этапе алгоритма решето Аткина представляет собой массив A размером n, заполненный нулями. Для определения простых чисел перебираются все $x, y < \sqrt{n}$. Для каждой такой пары вычисляется $4x^2 + y^2$, $3x^2 + y^2$, $3x^2 - y^2$ и значение элементов массива $A[4x^2 + y^2]$, $A[3x^2 + y^2]$, $A[3x^2 - y^2]$ 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 $O(n)$. При использовании «wheel factorization» и сегментирования оценка сложности алгоритма снижается до $O(n/\log(\log(n)))$, а потребление памяти до $O(\sqrt{n})$.

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. 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 $M_p = 2^p - 1$ простое тогда и только тогда, когда p — простое и M_p делит нацело $(p-1)$-й член последовательности S_k задаваемой рекуррентно: $S_1 = 4$, $S_k = S_{k-1}^2 - 2$ для $k > 1$.

Для числа M_p длиной p бит вычислительная сложность алгоритма составляет $O(p^3)$.

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.

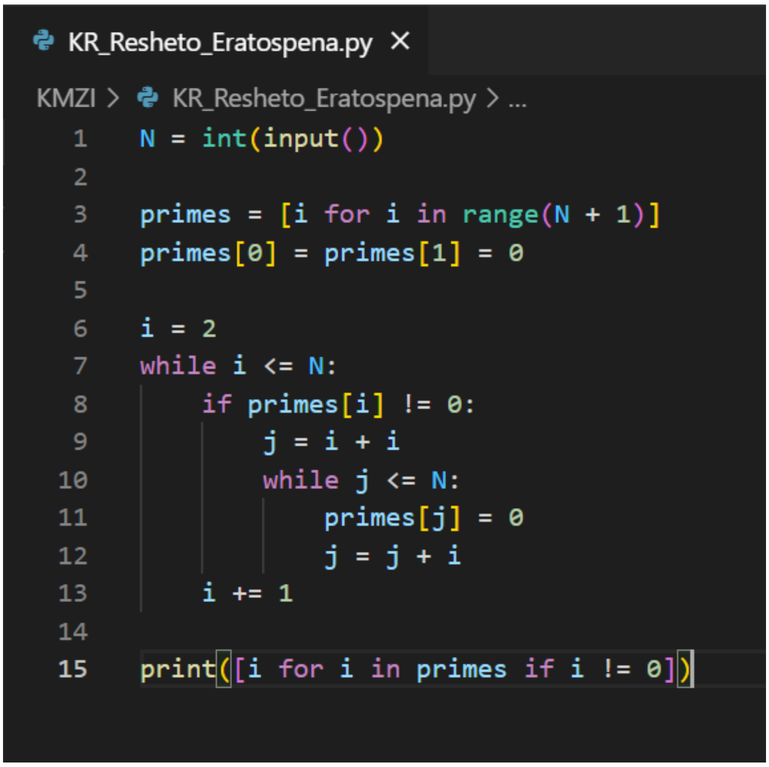

Рисунок 2 – Код программы (Решето Эратосфена)

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".

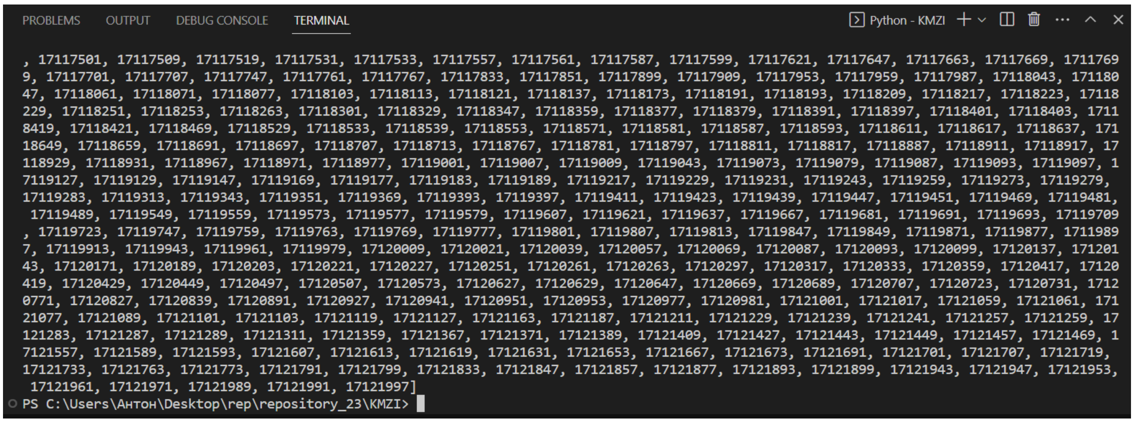

Рисунок 3 – Результат работы программы (Решето Эратосфена)

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 $a_1, a_2, \dots, a_k, k > 2$ 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, c), где c — 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: $220=2\cdot2\cdot5\cdot11$ и $600=2\cdot2\cdot2\cdot3\cdot5\cdot5$. Общими в этих двух произведениях будут множители 2,2 и 5. Это значит, что НОД(220, 600)=$2\cdot2\cdot5=20$.

2.2. Реализация метода проверки чисел на взаимную простоту

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

```
KR_Evklidov_Algoritm.py ×
KMZI > KR_Evklidov_Algoritm.py > ...
a = int(input("Введите первое число: "))
b = int(input("Введите второе число: "))

while a != b:
        if a >= b:
            a -= b
        else:
            b -= a

if a == 1:
    print("Числа взаимно простые!")
else:
    print("Числа не взаимно простые!")
```

Рисунок 4 – Код программы (Алгоритм Евклида)

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

```
PS C:\Users\Антон\Desktop\rep\repository_23\KMZI> &
R_Evklidov_Algoritm.py
Введите первое число: 322002
Введите второе число: 17121997
Числа взаимно простые!
PS C:\Users\Антон\Desktop\rep\repository_23\KMZI>
```

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

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.

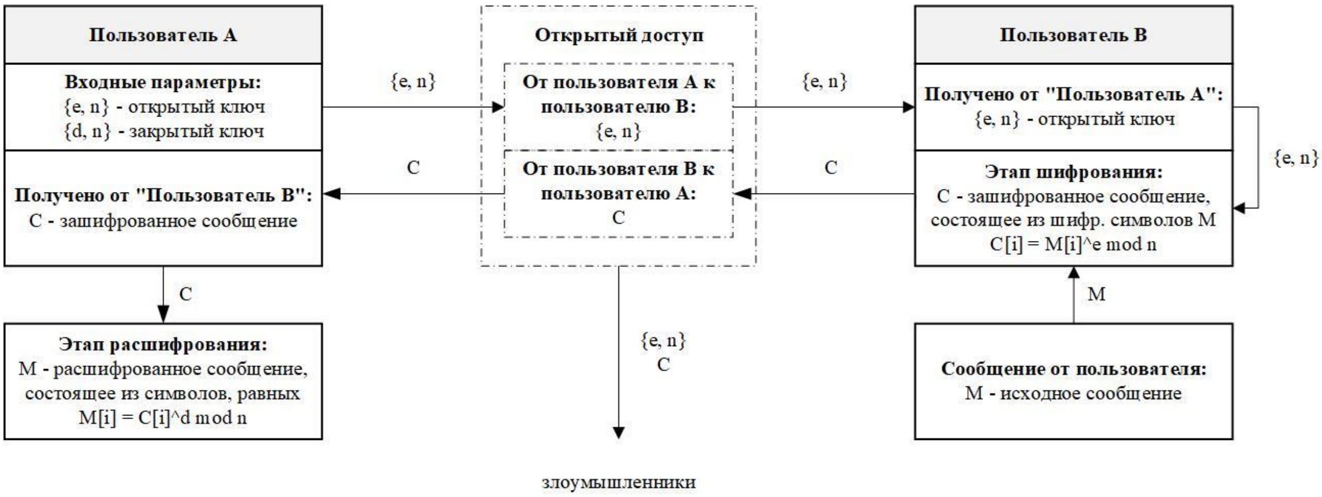

Рисунок 6 – Шифрование и расшифрование методом RSA

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

$$C[i] = (M[i]^e)\ mod\ n \quad (1)$$

В результате зашифрованное сообщение C отправляется обратно к "Пользователь А".

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

$$M[i] = (C[i]^d)\ mod\ n \quad (2)$$

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

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.

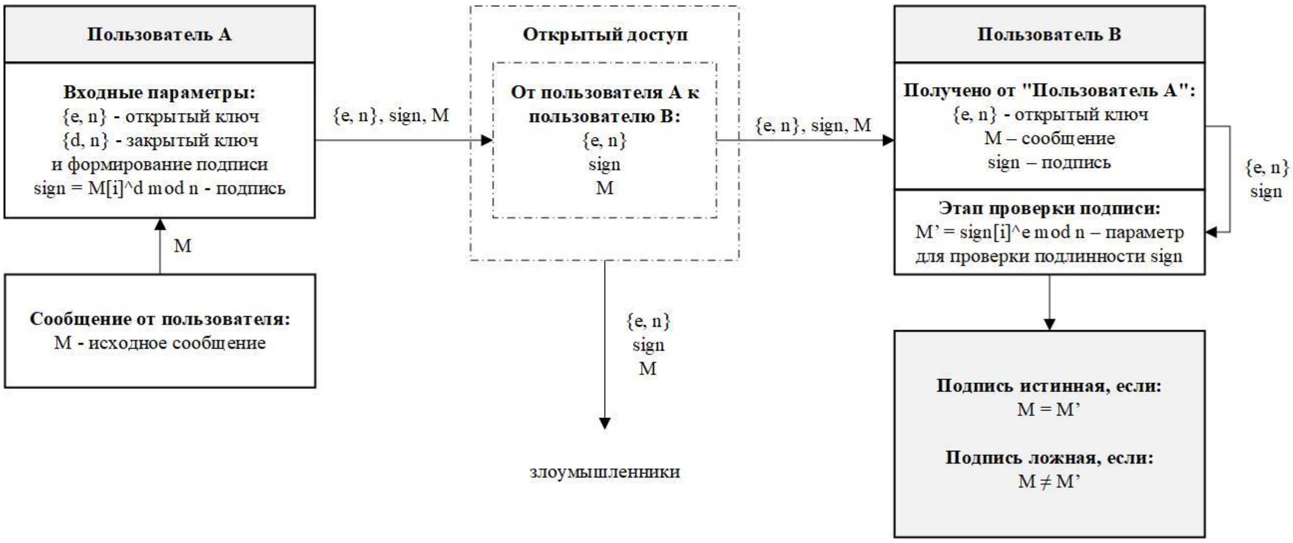

Рисунок 7 – Формирование и проверка подлинности ЭЦП методом RSA

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):

$$sign = (M[i]^{d})\ mod\ n \qquad (3)$$

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:

$$M' = (sign[i]^{e})\ mod\ n \qquad (4)$$

В случае, если $M = M'$ - подпись прошла проверку, то есть она истинная. Иначе, если $M \neq M'$ - 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.

p	q	n	Φ(n)	Открытый ключ		Секретный ключ	
		(p)(q)	(p-1)(q-1)	e	n	d	n
3	11	33	20	3	33	7	33

Выбор значения e	
<Φ	НОД
1	1
2	2
3	1
4	4
5	5
6	2
7	1
8	4
9	1
10	10
11	1
12	4
13	1
14	2
15	5
16	4
17	1
18	2
19	1

Выбор значения d		
d	e	(d*e)%Φ = 1
1	3	3
2	3	6
3	3	9
4	3	12
5	3	15
6	3	18
7	3	1
8	3	4
9	3	7
10	3	10
11	3	13
12	3	16
13	3	19
14	3	2
15	3	5
16	3	8
17	3	11
18	3	14
19	3	17

Исходное сообщение M	Зашифрованное сообщ. C	Расшифрованное сообщ. D
	M^e%n	C^d%n
15	9	15

Исходное сообщение M	Зашифрованное сообщ. C	Расшифрованное сообщ. D
	M^e%n	C^d%n
20	14	20

Рисунок 8 – 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).

Исходное сообщение M	Зашифрованное сообщ. C	Расшифрованное сообщ. D
	M^e%n	C^d%n
15	9	15

Проверка ЭЦП		
M	sign	M'
15	27	15
Подпись истинная		

Исходное сообщение M	Зашифрованное сообщ. C	Расшифрованное сообщ. D
	M^e%n	C^d%n
20	14	20

Проверка ЭЦП		
M	sign	M'
20	26	20
Подпись истинная		

Рисунок 9 – 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 М' 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 М:', 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.).

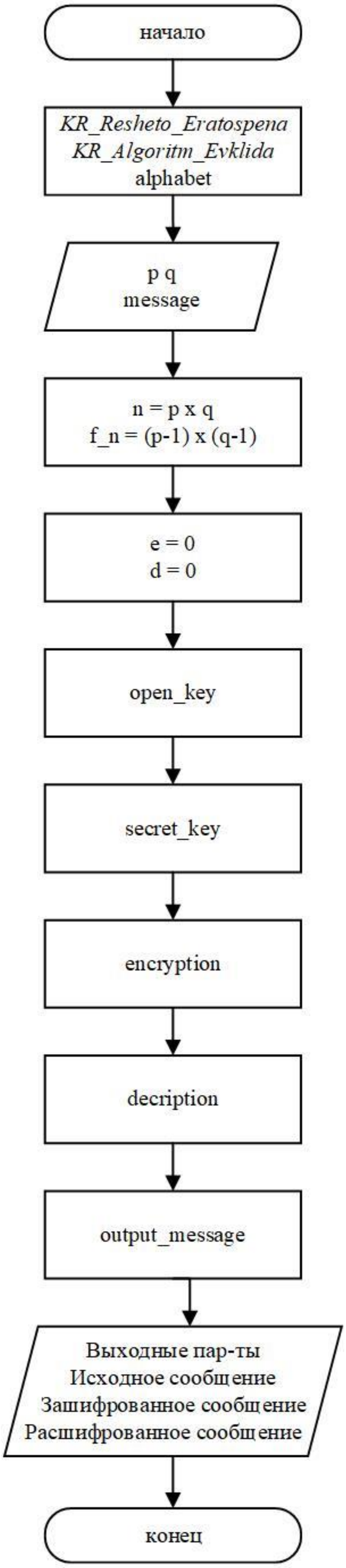

Рисунок 10 – Блок схема основной программы

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).

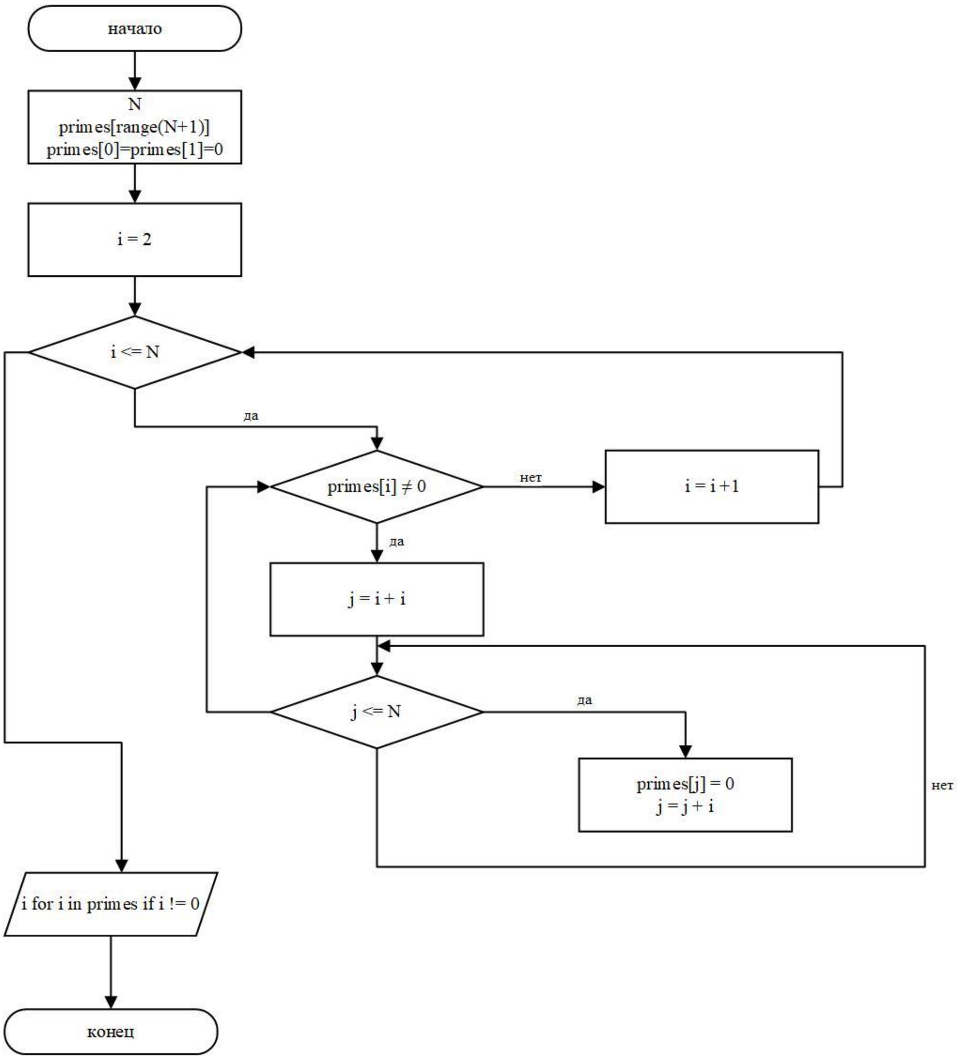

Рисунок 11 – Блок схема (Решето Эратосфена)

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.

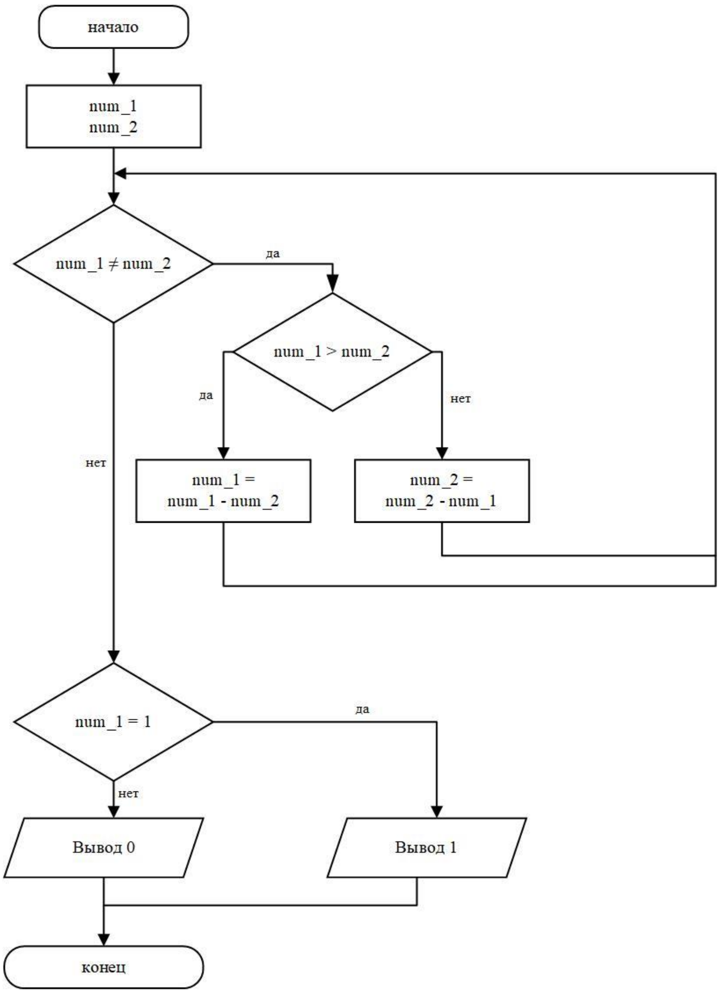

Рисунок 12 – Блок схема (Алгоритм Евклида)

4.2.4. Результат работы программы

На рисунке 13 представлен результат работы программы.

```
Введите число p: 997
Введите число q: 97
Введите сообщение, которое нужно зашифровать: курсовая работа

n: 96709
e: 5
d: 76493

        Исходное сообщение:
курсовая работа
[1082, 1091, 1088, 1089, 1086, 1074, 1072, 1103, 32, 1088, 1072, 1073, 1086, 1090, 1072]
Двоичный вид:
['0b10000111010', '0b10001000011', '0b10001000000', '0b10001000001', '0b10000111110', '0b10000110010', '0b10000110000', '0b10001001111', '0
b100000', '0b10001000000', '0b10000110000', '0b10000110001', '0b10000111110', '0b10001000010', '0b10000110000']

        Зашифрованное сообщение:
鵖滀桼� �昷� 볼�桼� 䶌�艺�
[40217, 28509, 16697, 94888, 76804, 20881, 96148, 47204, 93118, 16697, 96148, 45703, 76804, 20084, 96148]
Двоичный вид:
['0b1001110100011001', '0b110111101011101', '0b100000100111001', '0b10111001010101000', '0b10010110000000100', '0b101000110010001', '0b1011
1011110010100', '0b1011100001100100', '0b10110101110111110', '0b100000100111001', '0b10111011110010100', '0b1011001010000111', '0b100101100
00000100', '0b100111001110100', '0b10111011110010100']

        Расшифрованное сообщение:
курсовая работа
[1082, 1091, 1088, 1089, 1086, 1074, 1072, 1103, 32, 1088, 1072, 1073, 1086, 1090, 1072]
Двоичный вид:
['0b10000111010', '0b10001000011', '0b10001000000', '0b10001000001', '0b10000111110', '0b10000110010', '0b10000110000', '0b10001001111', '0
b100000', '0b10001000000', '0b10000110000', '0b10000110001', '0b10000111110', '0b10001000010', '0b10000110000']

        Создание ЭЦП
Сообщение M: курсовая работа [1082, 1091, 1088, 1089, 1086, 1074, 1072, 1103, 32, 1088, 1072, 1073, 1086, 1090, 1072]
sign: [34860, 11049, 66547, 74227, 28667, 1736, 45419, 4233, 2, 66547, 45419, 30959, 28667, 85929, 45419]
M': [1082, 1091, 1088, 1089, 1086, 1074, 1072, 1103, 32, 1088, 1072, 1073, 1086, 1090, 1072]
Подпись истинная
```

Рисунок 13 – 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.