1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский политехнический университет Петра Великого
3. —
4. Институт компьютерных наук и технологий
5. **Кафедра «Информационная безопасность компьютерных систем»**

**ОТЧЕТ**

**по лабораторной работе № 1**

1. по дисциплине «Теоретико-числовые методы в криптографии»
2. Вариант 36
3. Выполнил
4. студент гр. 33508/3 Проценко Е.Г.

\_\_\_\_\_\_\_\_\_\_

Руководитель Павленко Е.Ю.

\_\_\_\_\_\_\_\_\_\_

1. Санкт-Петербург
2. 2016

1. Содержание

[1 Цель работы 3](#_Toc463208460)

[2 Теоретические сведения 4](#_Toc463208461)

[3 Результаты работы 5](#_Toc463208462)

[3.1 Первая пара чисел 5](#_Toc463208463)

[3.1.1 Расширенный алгоритм Евклида 5](#_Toc463208464)

[3.1.2 Расширенный бинарный алгоритм Евклида 6](#_Toc463208465)

[3.1.3 Расширенный алгоритм Евклида с «усеченными» остатками 7](#_Toc463208466)

[3.2 Вторая пара чисел 8](#_Toc463208467)

[3.2.1 Расширенный алгоритм Евклида 8](#_Toc463208468)

[3.2.2 Расширенный бинарный алгоритм Евклида 9](#_Toc463208469)

[3.2.3 Расширенный алгоритм Евклида с «усеченными» остатками 10](#_Toc463208470)

[3.3 Третья пара чисел 11](#_Toc463208471)

[3.3.1 Расширенный алгоритм Евклида 11](#_Toc463208472)

[3.3.2 Расширенный бинарный алгоритм Евклида 12](#_Toc463208473)

[3.3.3 Расширенный алгоритм Евклида с «усеченными» остатками 14](#_Toc463208474)

[3.4 Линейное представление 15](#_Toc463208475)

[4 Вывод 17](#_Toc463208476)

[Список используемых источников 18](#_Toc463208477)

[Приложение А 19](#_Toc463208478)

[Приложение Б 20](#_Toc463208479)

[Приложение В 21](#_Toc463208480)

# Цель работы

Для каждой пары чисел найти наибольший общий делитель и его линейное представление, используя следующие алгоритмы:

1. Расширенный алгоритм Евклида.
2. Расширенный бинарный алгоритм Евклида.
3. Расширенный алгоритм Евклида с «усечёнными» остатками.

В отчете привести все итерации алгоритмов (последовательность остатков и две последовательности коэффициентов линейного представления). При числе итераций больше 20 привести первые 5 и последние 5 элементов каждой последовательности. Итерации должны быть пронумерованы.

При получении различных линейных представлений одного и того же наибольшего общего делителя обосновать корректность полученных результатов.

В отчете привести сравнение быстродействия реализованных алгоритмов.

Пары чисел приведены в Приложении А.

# Теоретические сведения

**Алгоритм Евклида** – эффективный алгоритм для нахождения наибольшего общего делителя двух целых чисел.

Грубая оценка сложности алгоритма: . Т.к. самая сложная операция – остаток от деления, а она приравнивается по сложности к операции умножения.

Расширенный алгоритм так же находит коэффициенты линейного представления:

Здесь x, y – целые коэффициенты линейного представления.

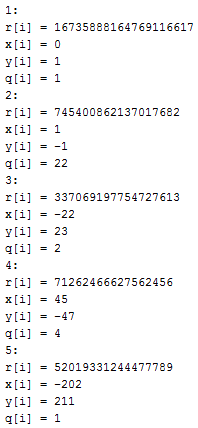
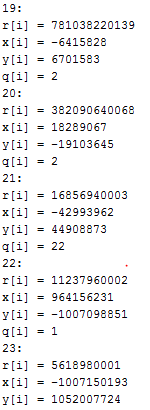
**Бинарный алгоритм Евклида –** по идее должен быть быстрее для реализации на компьютере, поскольку этот алгоритм использует такие особенности как побитовый сдвиг и основывается на следующих двух свойствах:

**Алгоритм Евклида с усеченными остатками –** позволяет избежать некоторого количества итераций, хотя сложность вычислений остается прежней.

# Результаты работы

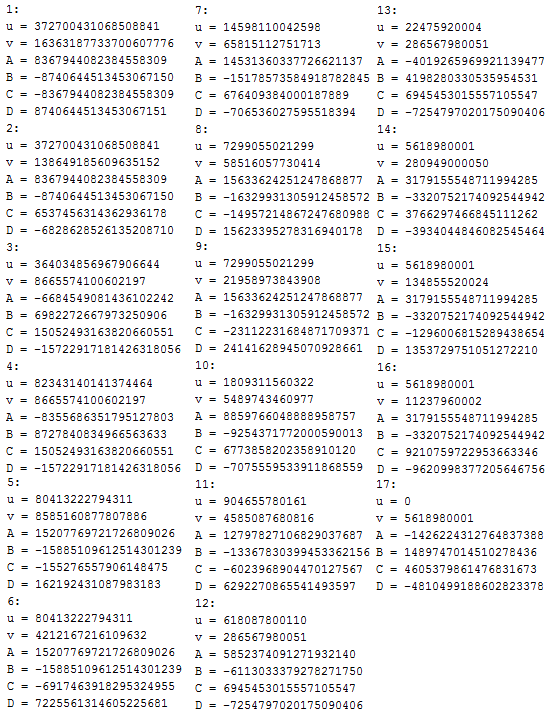
## Первая пара чисел

### Расширенный алгоритм Евклида

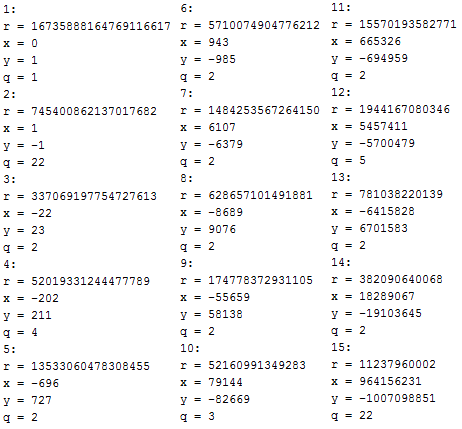
*Рисунок 1 – Первые 5 и крайние 5 итераций расширенного алгоритма Евклида для первой пары чисел*

### Расширенный бинарный алгоритм Евклида



*Рисунок 2 – Все итерации расширенного бинарного алгоритма Евклида для первой пары чисел*

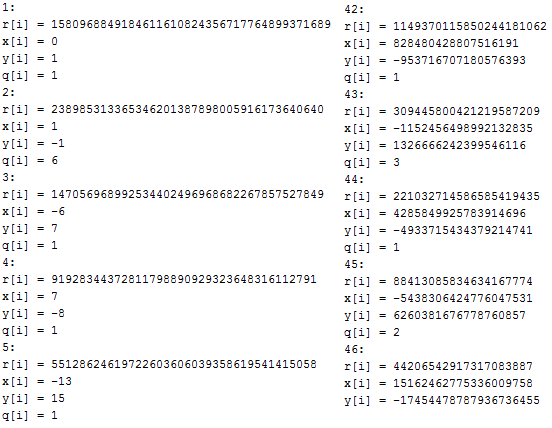
### Расширенный алгоритм Евклида с «усеченными» остатками



*Рисунок 3 – Все итерации расширенного алгоритма Евклида с «усеченными» остатками для первой пары чисел*

## Вторая пара чисел

### Расширенный алгоритм Евклида



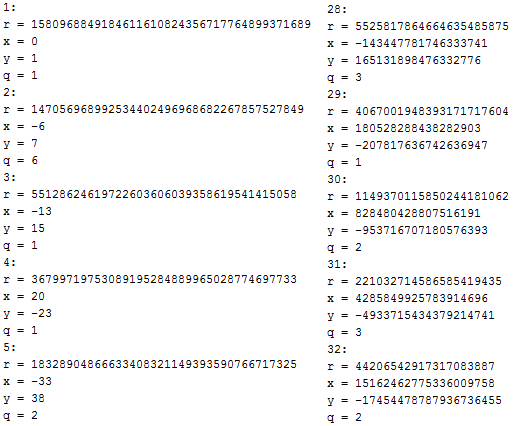
*Рисунок 4 – Первые 5 и крайние 5 итераций расширенного алгоритма Евклида для второй пары чисел*

### Расширенный бинарный алгоритм Евклида



*Рисунок 5 – Первые 5 и крайние 5 итераций расширенного бинарного алгоритма Евклида для второй пары чисел*

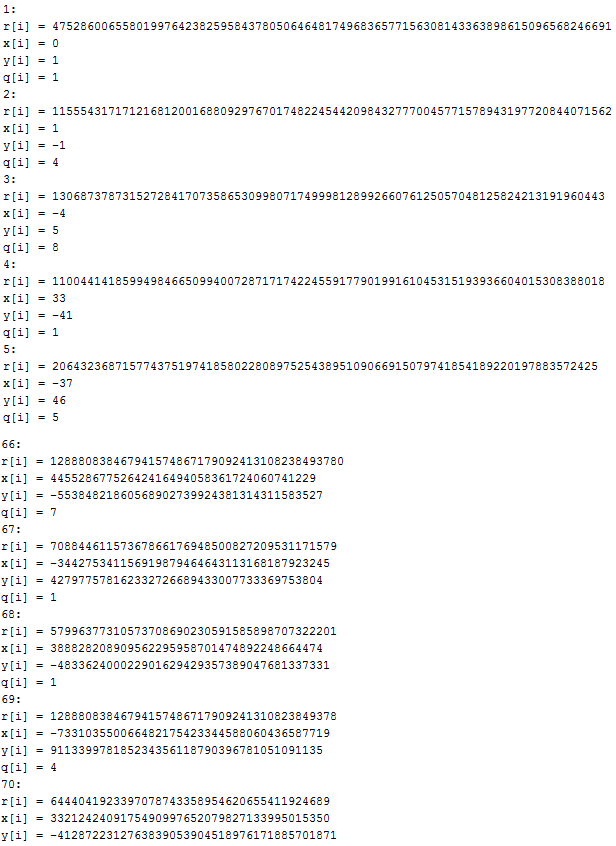
### Расширенный алгоритм Евклида с «усеченными» остатками



*Рисунок 6 – Первые 5 и крайние 5 итераций расширенного алгоритма Евклида с «усеченными» остатками для второй пары чисел*

## Третья пара чисел

### Расширенный алгоритм Евклида



*Рисунок 7 – Первые 5 и крайние 5 итераций расширенного алгоритма Евклида для третьей пары чисел*

### Расширенный бинарный алгоритм Евклида

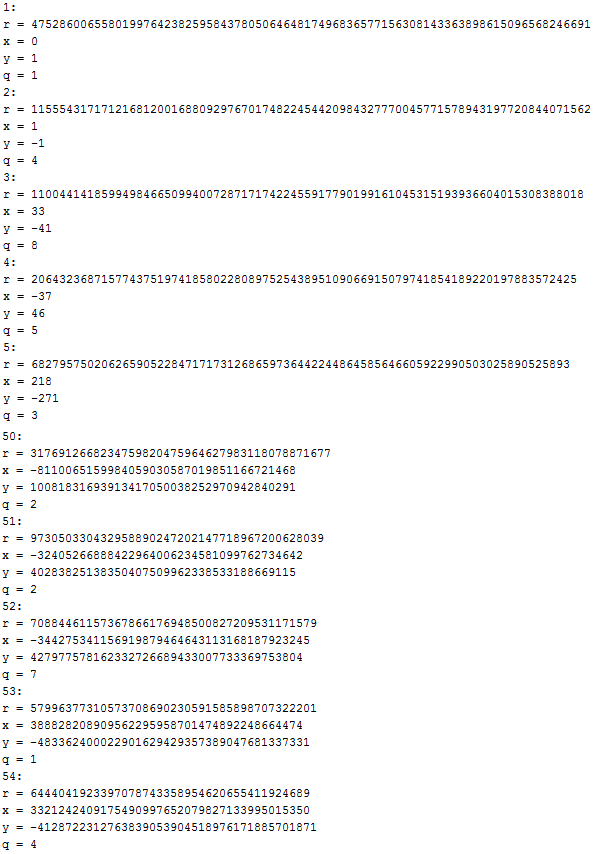


*Рисунок 8 – Первые 5 итераций расширенного бинарного алгоритма Евклида для третьей пары чисел*



*Рисунок 9 – Крайние 5 итераций расширенного бинарного алгоритма Евклида для третьей пары чисел*

### Расширенный алгоритм Евклида с «усеченными» остатками



*Рисунок 10 – Первые 5 и крайние 5 итераций расширенного алгоритма Евклида с «усеченными» остатками для второй пары чисел*

## Линейное представление

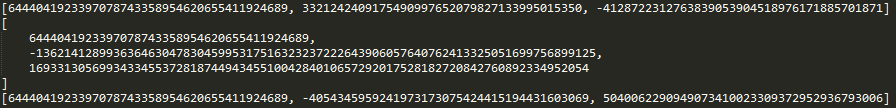
Далее будут приведены результаты работы трех алгоритмов. Первое число – НОД. Второе и третье – коэффициенты линейного представления. Первая строка – расширенный алгоритм Евклида. Вторая строка – расширенный бинарный алгоритм Евклида. Третья – расширенный алгоритм Евклида с «усеченными» остатками.



*Рисунок 11 – результаты работы алгоритмов для первой пары чисел*



*Рисунок 12 - результаты работы алгоритмов для второй пары чисел*



*Рисунок 13 - результаты работы алгоритмов для третьей пары чисел*

В разных случаях могут получаться разные коэффициенты x и y.

Рассмотрим 2 уравнения: и .

Вычитаем из одного другое, получаем: .

Нужно найти такие s, t, чтобы .

Пусть , . . .

s должно делаться на . .

Пусть , , .

Таким образом, пара чисел на ограничена одним линейным представлением

# Вывод

Это работа познакомила меня с очень интересной особенностью языка Python: в переменную можно записать очень большое число, почти сколь угодно большое.

Познакомился с алгоритмом нахождения НОД, думаю, это очень важный алгоритм, когда речь идет о криптографических алгоритмах.

Были проведены замеры количества времени, которое выполняется алгоритм и количество итераций для каждой пары чисел в каждом алгоритме. Результаты измерений моно увидеть в Приложении Б.

Алгоритм с «усеченными» остатками в теории должен был давать меньшее кол-во итераций и меньшее врем вычислений. Практический результат это подтверждает.

Однако бинарный алгоритм, который должен быстро считаться на компьютерах из-за операции сдвига, показал очень плохой результат. Возможно это связано с одной или всеми следующими утверждения:

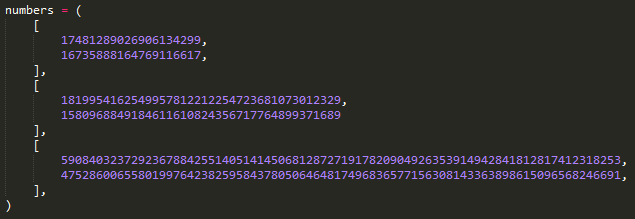
* На каждой итерации цикла приходится делать очень много проверок на делимость чисел на 2.
* Замедление алгоритма связано с особенностями языка Python, т.к. каждая переменная – это объект, а не просто число.
* Автор кода плохо написал алгоритм

Возможны и другие причины.

# Список используемых источников

1. «Теоретико-числовые методы в криптографии» Е.Б.Маховенко 2006

# Приложение А



*Рисунок 14 – Пары чисел, необходимые для демонстрации работы алгоритмов.*

# Приложение Б

Для каждой пары чисел и алгоритма для замера времени выполнения функция вызывалась 1000000 раз.

*Таблица 1 – Измерения работы алгоритмов*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Пара чисел | Параметр измерения | Расширенный алгоритм Евклида | Расширенный бинарный алгоритм Евклида | Расширенный алгоритм с “усеченными” остатками |
| 1 | Итерации алгоритма | 23 | 17 | 15 |
| Время | 39.943881057349735 | 56.822424654786715 | 36.2147915377467 |
| 2 | Итерации алгоритма | 46 | 42 | 32 |
| Время | 93.1904101076962 | 139.64993112626624 | 89.8772101711601 |
| 3 | Итерации алгоритма | 70 | 101 | 54 |
| Время | 165.00916334985078 | 330.95354647341253 | 160.6251639519881 |

# 

# Приложение В

Исходный код

**def** **setter**(a, b):

r = []

**if** a > b:

r.append(a)

r.append(b)

**else**:

r.append(b)

r.append(a)

**return** r

**def** **extended\_euclid**(a, b):

r = setter(a, b)

x = [**1**, **0**]

y = [**0**, **1**]

q = [**0**]

i = **1**

**while** True:

# calculating new iteration

r.append(r[i - **1**] % r[i])

# checking if nod is found

**if** r[i + **1**] == **0**:

**break**

**else**:

# calculating new iteration

q.append(int(r[i - **1**] / r[i]))

x.append(x[i - **1**] - x[i] \* q[i])

y.append(y[i - **1**] - y[i] \* q[i])

i += **1**

**return** [r[i], x[i], y[i]]

**def** **extended\_cut\_euclid**(a, b):

**def** **swapper**():

nonlocal r

nonlocal x

nonlocal y

tmp = x[**0**]

x[**0**] = x[**1**]

x[**1**] = tmp

tmp = y[**0**]

y[**0**] = y[**1**]

y[**1**] = tmp

tmp = r[**0**]

r[**0**] = r[**1**]

r[**1**] = tmp

r = setter(a, b)

a = r[**0**]

b = r[**1**]

x = [**1**, **0**]

y = [**0**, **1**]

**while** True:

q = int(r[**0**] / r[**1**])

r[**0**] = r[**0**] % r[**1**]

**if** r[**0**] == **0**:

d = r[**1**]

**return** [d, x[**1**], y[**1**]]

x[**0**] -= q \* x[**1**]

y[**0**] -= q \* y[**1**]

**if** abs(r[**0**]) > abs(r[**1**] / **2**):

r[**1**] -= r[**0**]

x[**1**] -= x[**0**]

y[**1**] -= y[**0**]

**else**:

swapper()

**def** **extended\_binary\_euclid**(a, b):

**def** **is\_even**(x):

bit = **0x1**

**return** x & bit == **0**

r = setter(a, b)

a = r[**0**]

b = r[**1**]

# step 1

g = **1**

# step 2

**while** is\_even(a) **and** is\_even(b):

a >>= **1**

b >>= **1**

g <<= **1**

# step 3

u = a

v = b

A = **1**

B = **0**

C = **0**

D = **1**

# step 4

**while** u != **0**:

# step 4.1

**while** is\_even(u):

# step 4.1.1

u >>= **1**

# step 4.1.2

**if** is\_even(A) **and** is\_even(B):

A >>= **1**

B >>= **1**

**else**:

A = (A + b) >> **1**

B = (B - a) >> **1**

# step 4.2

**while** is\_even(v):

# step 4.2.1

v >>= **1**

# step 4.2.2

**if** is\_even(C) **and** is\_even(D):

C >>= **1**

D >>= **1**

**else**:

C = (C + b) >> **1**

D = (D - a) >> **1**

# step 4.3

**if** u >= v:

u -= v

A -= C

B -= D

**else**:

v -= u

C -= A

D -= B

# step 5

d = g \* v

z = C

x = D

# step 6

**return** [d, z, x]

numbers = (

[

**17481289026906134299**,

**16735888164769116617**,

],

[

**1819954162549957812212254723681073012329**,

**1580968849184611610824356717764899371689**,

],

[

**59084032372923678842551405141450681287271917820904926353914942841812817412318253**,

**47528600655801997642382595843780506464817496836577156308143363898615096568246691**,

],

)