|  |  |
| --- | --- |
| **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](#_Toc131078123)

[Исходные данные 4](#_Toc131078124)

[1. Вычисление простого числа 5](#_Toc131078125)

[1.1. Теоретическая часть 5](#_Toc131078126)

[1.1.1. Решето Эратосфена 5](#_Toc131078127)

[1.1.2. Решето Аткина 6](#_Toc131078128)

[1.1.3. Числа Марсенна и тест Люка-Лемера 7](#_Toc131078129)

[1.1.4. Теорема Ферма и тест Миллера-Рабина 7](#_Toc131078130)

[1.1.5. Тест Миллера-Рабина 8](#_Toc131078131)

[1.2. Реазилация методов 9](#_Toc131078132)

[1.3. Проверка 9](#_Toc131078133)

[2. Проверка чисел на взаимную простоту 9](#_Toc131078134)

[2.1. Теоретическая часть 9](#_Toc131078135)

[2.1.1. Взаимно простые числа 9](#_Toc131078136)

[2.1.2. Алгоритм Евклида 11](#_Toc131078137)

[2.1.3. Разложение на множители 11](#_Toc131078138)

[2.2. Реализация методов 12](#_Toc131078139)

[2.3. Формирование групп взаимно простых чисел 12](#_Toc131078140)

[3. Метод RSA 12](#_Toc131078141)

[4. Реализация метода RSA 14](#_Toc131078142)

[4.1. Реализация метода в Эксель 14](#_Toc131078143)

[4.2. Реализация методов с помощью программы 14](#_Toc131078144)

[5. Область применения метода RSA 14](#_Toc131078145)

[6. Заключение 14](#_Toc131078146)

# 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 Эксель и программно)

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 метод 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. Этот метод позволяет найти все простые числа меньше заданного числа *n*. Суть метода заключается в следующем. Возьмем набор чисел от 2 до *n*. Вычеркнем из набора (отсеим) все числа делящиеся на 2, кроме 2. Перейдем к следующему «не отсеянному» числу — 3, снова вычеркиваем все что делится на 3. Переходим к следующему оставшемуся числу — 5 и так далее до тех пор пока мы не дойдем до *n*. После выполнения вышеописанных действий, в изначальном списке останутся только простые числа.

Алгоритм можно несколько 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.

Существует ряд оптимизаций, позволяющих снизить эти показатели. Прием под названием «[wheel factorization](https://en.wikipedia.org/wiki/Wheel_factorization)» состоит в том, чтобы включать в изначальный список только числа [взаимно простые](https://ru.wikipedia.org/wiki/%D0%92%D0%B7%D0%B0%D0%B8%D0%BC%D0%BD%D0%BE_%D0%BF%D1%80%D0%BE%D1%81%D1%82%D1%8B%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%B0) с несколькими первыми простыми числами (например меньше 30). В теории предлагается брать первые простые примерно до. Это позволяет снизить сложность алгоритма в раз. Помимо этого для уменьшения потребляемой памяти используется так называемое сегментирование. Изначальный набор чисел делится на сегменты размером и для каждого сегмента решето Эратосфена применяется по отдельности. Потребление памяти снижается до .

### 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 .

### 1.1.4. Теорема Ферма и тест Миллера-Рабина

Простых чисел Мерсенна известно не очень много, поэтому для криптографии с открытым ключом необходим другой способ поиска простых чисел. Одним из таким способов является [тест простоты Ферма](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82_%D0%A4%D0%B5%D1%80%D0%BC%D0%B0). Он основан на малой теореме Ферма, которая гласит, что если *n* — простое число, то для любого *a*, которое не делится на *n*, выполняется равенство .

Тест простоты Ферма — вероятностный тест, который заключается в переборе нескольких значений *a*, если хотя бы для одного из них выполняется неравенство , то число *n* — составное. В противном случае, *n* — вероятно простое. Чем больше значений *a* использовано в тесте, тем выше вероятность того, что *n* — простое.

К сожалению, существуют такие составные числа *n*, для которых сравнение  выполняется для всех *a* взаимно простых с *n*. Такие числа называются [числам Кармайкла](https://ru.wikipedia.org/wiki/%D0%A7%D0%B8%D1%81%D0%BB%D0%BE_%D0%9A%D0%B0%D1%80%D0%BC%D0%B0%D0%B9%D0%BA%D0%BB%D0%B0). Составные числа, которые успешно проходят тест Ферма, называются псевдопростыми Ферма. Количество псевдопростых Ферма бесконечно, поэтому тест Ферма — не самый надежный способ определения простых чисел.

### 1.1.5. Тест Миллера-Рабина

Более надежных результатов можно добиться комбинируя малую теорему Ферма и тот факт, что для простого числа *p* не существует других корней уравнения , кроме 1 и -1. Тест Миллера-Рабина перебирает несколько значений *a* и проверяет выполнение следующих условий.

Пусть *p* – простое число и , тогда для любого *a* справедливо хотя бы одно из условий:

1. Существует целое число *r < s* такое, что

По теореме Ферма , а так как  из свойства о корнях уравнения  следует что если мы найдем такое *a*, для которого одно из условий не выполняется, значит *p* — составное число. Если одно из условий выполняется, число *a* называют свидетелем простоты числа *n* по Миллеру, а само число *n* — вероятно простым.

Чем больше свидетелей простоты найдено, тем выше вероятность того, что *n* – простое. Согласно теореме Рабина вероятность того, что случайно выбранное число *a* окажется свидетелем простоты составного числа составляет приблизительно 1/4.

Следовательно, если проверить *k* случайных чисел *a*, то вероятность принять составное число за простое .

Сложность работы алгоритма , где *k* — количество проверок.

Благодаря быстроте и высокой точности тест Миллера-Рабина широко используется при поиске простых чисел. Многие современные криптографические библиотеки при проверке больших чисел на простоту используют только этот тест.

## 1.2. Реазилация методов

## 1.3. Проверка

# 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 методов

## 2.3. Формирование групп взаимно простых чисел

# 3. Метод RSA

Шифрование RSA - это система, которая решает то, что когда-то было одной из самых больших проблем в криптографии: Как вы можете отправить кому-то закодированное сообщение не имея возможности ранее поделиться кодом с ними?

Это одна из фундаментальных проблем криптографии, которая решается схемы шифрования с открытым ключом (также известные как асимметричное шифрование), такие как RSA.

При шифровании RSA сообщения шифруются с помощью кода, называемого открытый ключ, которыми можно поделиться открыто. Из-за некоторых четких математических свойств алгоритма RSA, если сообщение было зашифровано открытым ключом, оно может быть расшифровано только другим ключом, известным как закрытый ключ. У каждого пользователя RSA есть пара ключей, состоящая из их открытого и закрытого ключей. Как следует из названия, закрытый ключ должен храниться в секрете.

Схемы шифрования с открытым ключом отличаются от шифрование с симметричным ключом, где и в процессе шифрования, и в дешифровании используется один и тот же закрытый ключ. Эти различия делают шифрование с открытым ключом, такое как RSA, полезным для связи в ситуациях, когда не было возможности безопасно распространять ключи заранее.

Алгоритмы с симметричным ключом имеют свои собственные приложения, такие как шифрование данных для личного использования или для случаев, когда существуют защищенные каналы, по которым закрытые ключи могут совместно использоваться.

RSA-шифрование часто используется в комбинация с другими схемами шифрования, или для цифровые подписи который может доказать подлинность и целостность сообщения. Обычно он не используется для шифрования целых сообщений или файлов, потому что он менее эффективен и требует больше ресурсов, чем шифрование с симметричным ключом.

Чтобы сделать вещи более эффективными, файл обычно шифруется с помощью алгоритма симметричного ключа, а затем симметричный ключ шифруется с помощью шифрования RSA.. В рамках этого процесса только объект, имеющий доступ к закрытому ключу RSA, сможет расшифровать симметричный ключ.

Не имея доступа к симметричному ключу, исходный файл не может быть расшифрован. Этот метод можно использовать для обеспечения безопасности сообщений и файлов, не занимая слишком много времени и не занимая слишком много вычислительных ресурсов.

Являясь одной из первых широко используемых схем шифрования с открытым ключом, RSA заложила основы для большей части наших безопасных коммуникаций. это было традиционно используется в TLS и был также оригинальный алгоритм, используемый в шифровании PGP. RSA по-прежнему используется в различных веб-браузерах, электронной почте, VPN, чатах и ​​других каналах связи.

# 4. Реализация метода RSA

## 4.1. Реализация метода в Эксель

## 4.2. Реализация методов с помощью программы

# 5. Область применения метода RSA

# 6. Заключение