СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#__RefHeading___1)

[1. Описание проекта 4](#__RefHeading___2)

[1.1. Описание RSA 4](#__RefHeading___3)

[1.2. Постановка задачи 5](#__RefHeading___4)

[1.3. Используемые инструменты. 5](#__RefHeading___5)

[2. Ход работы. 5](#__RefHeading___6)

[2.1. Тест Миллера - Рабина. 6](#__RefHeading___7)

[2.2. Расширенный алгоритм Евклида 7](#__RefHeading___8)

[2.3. Метод быстрого возведения в степень 7](#__RefHeading___9)

[3. Реализация проекта 8](#__RefHeading___10)

[3.1. Сложности в разработке 8](#__RefHeading___11)

[3.2. Полученный результат 8](#__RefHeading___12)

[ЗАКЛЮЧЕНИЕ 9](#__RefHeading___13)

[СПИСОК ЛИТЕРАТУРЫ 11](#__RefHeading___14)

[ПРИЛОЖЕНИЕ 12](#__RefHeading___15)

ВВЕДЕНИЕ

Проектно-технологическая практика проходила на кафедре системного анализа и информационных технологий Института вычислительной математики и информационных технологий КФУ с 09 февраля 2022 по 07 июня 2022 года. Целью практики является получение компетенций в области информационной безопасности и криптографии, в том числе изучение основных теоретико-числовых и криптографических алгоритмов. В данной работе будет рассмотрено шифрование и дешифрование RSA.

Актуальность: Актуальностью проблемы шифрования данных в сфере криптографии является то, что использование систем шифрования в сфере защиты информации велико и на сегодня существует множество различных алгоритмов, позволяющих осуществлять шифрование. Главным критерием каждого метода является его криптостойкость.

Предмет исследования:Процесс шифрования данных с использованием криптосистемы.

Предметы использования:

* Алфавит; конечное множество используемых для кодирования информации знаков.
* Текст; упорядоченный набор из элементов алфавита.
* Шифрование преобразовательный процесс; исходный текст, который носит также название открытого текста, заменяется шифрованным текстом.
* Дешифрование; обратный шифрованию процесс. На основе ключа шифрованный текст преобразуется в исходный.
* Ключ; информация, необходимая для беспрепятственного шифрования и дешифрования текстов.

# Описание проекта

## Описание RSA

В современном мире люди каждый день сталкиваемся с проблемой  
передачи информации через технологии коммуникации, ведь "голубиная почта" и "гонцы" остались далеко в прошлом, поэтому практически весь поток информации проходит через интернет, однако всегда найдутся те, кто захочет перехватить наши данные, поэтому с развитием информационных технологий люди так же были обязаны и защищать каналы связи, именно для этого создали алгоритмы шифрования, одним из которых является RSA

RSA - алгоритм шифрования один из самых простейших в понимании способов кодировки каких-либо данных, но при всем при этом весьма эффективный.

Распишем общую схему определения параметров:

1. Генерация двух случайных простых чисел **р** и **q**, заданного размера

2. Вычисляется модуль

3. Нахождение значения функции Эйлера

4. Выбирается открытый ключ е, удовлетворяющий следующей системе:

1)

2)

5. На основе выше указанных пунктов вычисляется секретный ключ d, мультипликативно обратное к числу e:

Казалось бы, в чем проблема расшифрования, ведь d так или иначе зависит от известных открытых ключей. Дело в том, что при отсутствии закрытого ключа, RSA можно рассматривать в качестве односторонней функции.

Пусть имеется некая F(x), зная ее аргумент, мы можем найти ее значение, однако противоположное действие неисполнимо, в этом и заключается смысл односторонней функции. Чтобы убедиться в этом, рассмотрим кодировку и расшифровку сообщений.

Рассмотрим пример зашифровки 1-го символа:

Пусть имеется сообщение единичной длины а, а также открытый ключ(e, n). Тогда наш зашифрованный символ будет вычисляться следующим образом:

После идет процесс расшифрования: в нашем арсенале имеется только значение числа п, однако для поиска закрытого ключа, т. е. пары {d, n} необходимо знать значения функции Эйлера,то есть стоит проблема факторизации числа, что является весьма сложным процессом, если п весьма большое . Это и есть основная проблема в получении закрытой экспоненты.Однако, если все же она была найдена, расшифровка происходит так:

## Постановка задачи

1. Узнать больше о Криптографии.

2. Узнать, что такое RSA.

3. Изучить расширенный алгоритм Евклида

4. Понять и изучить метод шифрования информации

5. Реализовать метод RSA.

## Используемые инструменты.

1. Среда разработки VS Code.

2. Язык программирования Python.

# Ход работы.

В начале процесса реализации программы были реализованы методы, которые будут необходимы в процессе разработки.

## Тест Миллера - Рабина.

Тест Миллера - Рабина - вероятностный полиномиальный тест простоты. Тест Миллера - Рабина даёт нам возможность проверить, является ли данное число составным.

Алгоритм был разработан Гари Миллером в 1976 году и модифицирован Майклом Рабином в 1980 году.

Пусть есть число *р*, которое надо проверить на простоту. Тогда проверка того, что *р - 1* делится на 2 работает за *Ologp*(длина числа р). Так как максимальная степень 2, на которую делится *р- 1* это ***l****ogp*, то за *0(log)р* можно представить так:

Далее можно взять случайное число *а* от 1 до *р*. С помощью быстрого возведения в степень можно посчитать по модулю *р*. Так как числа имеют длину *0* и умножение двух чисел работает за *0*, а значит быстрое возведение в степень работает за 0. Так было посчитано по модулю *р*. Если число сравнимо с *-1* или *1*, то а - свидетель простоты. Если же нет, то *n* раз возводим число в квадрат и проверяем, что не сравнимо ли оно *с - 1* по модулю *р*. Итого за можно проверить число на свидетеля простоты.

Дальше чем более точная вероятность того, что *р* составное вам нужна, тем больше раз вам надо повторить тест. При этом если повторить тест *к* раз каждое число свидетель простоты, то число составное с вероятностью

Алгоритм Миллера-Рабина параметризуется количеством раундов. Рекомендует брать порядка величины , где *m* - проверяемое число. Для данного *m* находятся такие целое число *S* и целое нечётное число *t*,что . Выбирается случайное число *а , 1<а<m*. Если *а* не является свидетелем простоты числа *m*, то выдается ответ «*m* составное», и алгоритм завершается. Иначе, выбирается новое случайное число *а* и процедура проверки повторяется. После нахождения *r* свидетелей простоты, выдается ответ «*m*, вероятно, простое», и алгоритм завершается .

## Расширенный алгоритм Евклида

Алгоритм основан на рекурсии, коэффициенты вычисляются на обратном ходу, то есть при возврате из рекурсии. Для этого используются следующие соотношения:

Пусть известны коэффициенты для пары *,* такие что:

, и нужно рассчитать коэффициенты *(x,y)* для нашей пары *(a,b)*, такие, что:

Раскроем выражение *b/a*:

,где

*[b/a]* – результат целочисленного деления b на a, и подставим в выражение:

выполнив перегруппировку слагаемых,получим:

Сравнивая это с исходным выражением над неизвестными *x* и *y*, получаем требуемые выражения:

Выход из рекурсии - конец обычного алгоритма, случай а = 0.НОД, соответственно, равен b, и требуемые коэффициенты х и у равны 0 и 1 соответственно. Дальше идет вычисление по формулам.

## Метод быстрого возведения в степень

Тут стоит заметить, что нам понадобится возводить большие числа в такие же по величине степени, поэтому без данного метода наша программа работала бы недопустимо медленно, поэтому был вынужден создать метод быстрого возведения в степень.

# Реализация проекта

В начале было принято решение реализовать методы алгоритмов Миллера-Рабина, быстрого возведения в степень, Евклида и расширенный алгоритм Евклида. Далее были найдены простые числа для вычисления модуля, используя алгоритм Миллера-Рабина, реализовал функцию Эйлера, вычислил открытую экспоненту, используя алгоритмы Евклида, нашёл закрытую экспоненту, используя расширенный алгоритм Евклида. После нахождения всех значений уже приступил к шифрованию и расшифрованию.

Шифрование происходит по следующим формулам:

Расшифрование происходит по следующим формулам:

Также при реализации проекта была добавлена возможность работы с файлами. Для шифрования и расшифрования нужно вписать нужное слово в файл и полученный результат также появится в файле.

## Сложности в разработке

Возникли проблемы с поиском простых чисел, в большинстве случаев число не являлось простым, однако решение оказалось довольно-таки легким, ошибка была обнаружена в количестве проведенных тестов.Данную проблему решил путем проведения 128 тестов при запуске программы.

## Полученный результат

В итоге, был реализован метод шифрования и дешифрования RSA. Были проведены тесты алгоритма, по этим тестам можно сказать, что алгоритм работает правильно.Также было реализовано индивидуальное задание, связанное с работой с файлами.

ЗАКЛЮЧЕНИЕ

Любые алгоритмы шифрования базируются на различных математических теоремах и различных аксиомах, так криптографическая система открытого ключа RSA является весьма комплексной структурой, так как включает в себя не менее сложные математические алгоритмы для эффективного и оптимального использования.

В данном проекте мы познакомились не только с задачей шифрования и расшифрования различного рода сообщений, мы научились вычислять наибольший общий делитель с помощью расширенного алгоритма Евклида, научились реализовывать быструю систему для возведения больших чисел в степени различной длины и последующего взятия модуля, которая значительно ускоряет процесс работы RSA. Более того, мы познакомились с такой сложной системой, как тест Миллера-Рабина, который позволяет с огромной точностью дать ответ на вопрос, является ли число простым или нет.

В процессе возникли трудности с простыми числами, не всегда число являлось простым, однако удалось исправить данный изъян, после чего код начал работать стабильно быстро

Так же стоит отметить важность RSA в современном мире, несмотря на то, что он был придуман давно, его и следы можно обнаружить во многих современных системах.

В результате учебной практики (технологической (проектно-технологической) практики) 4 семестра 2 курса были освоены следующие компетенции (таблица 1).

Таблица 1 - Компетенции

| Шифр компетенции | Расшифровка приобретаемой компетенции | Расшифровка освоения компетенции |
| --- | --- | --- |
| УК-6 | Способен применять компьютерные/суперкомпьютерные методы, современное программное обеспечение, в том числе отечественного происхождения, для решения задач профессиональной деятельности | Способен применять соврменные информационные технологии для решения прикладных задач |
| ПК-2 | Способен к разработке алгоритмических и программных решений в области системного и прикладного программирования, математических, информационных и имитационных моделей, созданию информационных ресурсов глобальных сетей, образовательного контента, прикладных баз данных, тестов и средств тестирования систем и средств на соответствие стандартам исходным требованиям | Способен к разработке алгоритмических решений в сфере информационной безопасности |

СПИСОК ЛИТЕРАТУРЫ

1. Брюс Шнайер. Прикладная криптография. - 2-е издание.Протоколы, алгоритмы и исходные тексты на языке С - 1040 с(дата обращения 19.05.2023).

2. Абелян В.3. Криптографический алгоритм RSA // International Journal of Advanced Studies in Computer Engineering, Nol, 2020. - 7 с.(дата обращения 19.05.2023).

3. Боревич 3.И. Теория чисел. / 3.И. Боревич, И.Р.Шафаревич. - 3-е издание, М.: Наука, 1985, 504 с. (дата обращения 19.05.2023).

4. Коутинхо С. Введение в теорию чисел. Алгоритм RSA. Москва: Постмаркет,2001,328 с.(дата обращения 19.05.2023).

5. Ященко В.В. Введение в криптографию.-4-е издание; Москва: МЦМНО,2012,348 с. (дата обращения 19.05.2023).

ПРИЛОЖЕНИЕ

import random

def rabinMiller(n, d):

a = random.randint(2, (n - 2) - 2)

x = pow(a, int(d), n) # a^d%n

if x == 1 or x == n - 1:

return True

# square x

while d != n - 1:

x = pow(x, 2, n)

d \*= 2

if x == 1:

return False

elif x == n - 1:

return True

# is not prime

return False

def isPrime(n):

"""

return True if n prime

fall back to rabinMiller if uncertain

"""

if n < 2:

return False

# low prime numbers to save time

lowPrimes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907, 911, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983, 991, 997]

# if in lowPrimes

if n in lowPrimes:

return True

# if low primes divide into n

for prime in lowPrimes:

if n % prime == 0:

return False

# find number c such that c \* 2 ^ r = n - 1

c = n - 1 # c even bc n not divisible by 2

while c % 2 == 0:

c /= 2 # make c odd

# prove not prime 128 times

for i in range(128):

if not rabinMiller(n, c):

return False

return True

def generateKeys(keysize=1024):

e = d = N = 0

# get prime nums, p & q

p = generateLargePrime(keysize)

q = generateLargePrime(keysize)

print(f"p: {p}")

print(f"q: {q}")

N = p \* q # RSA Modulus

phiN = (p - 1) \* (q - 1) # totient

# choose e

# e is coprime with phiN & 1 < e <= phiN

while True:

e = random.randrange(2 \*\* (keysize - 1), 2 \*\* keysize - 1)

if (isCoPrime(e, phiN)):

break

# choose d

# d is mod inv of e with respect to phiN, e \* d (mod phiN) = 1

d = modularInv(e, phiN)

return e, d, N

def generateLargePrime(keysize):

"""

return random large prime number of keysize bits in size

"""

while True:

num = random.randrange(2 \*\* (keysize - 1), 2 \*\* keysize - 1)

if (isPrime(num)):

return num

def isCoPrime(p, q):

"""

return True if gcd(p, q) is 1

relatively prime

"""

return gcd(p, q) == 1

def gcd(p, q):

"""

euclidean algorithm to find gcd of p and q

"""

while q:

p, q = q, p % q

return p

def egcd(a, b):

s = 0; old\_s = 1

t = 1; old\_t = 0

r = b; old\_r = a

while r != 0:

quotient = old\_r // r

old\_r, r = r, old\_r - quotient \* r

old\_s, s = s, old\_s - quotient \* s

old\_t, t = t, old\_t - quotient \* t

# return gcd, x, y

return old\_r, old\_s, old\_t

def modularInv(a, b):

gcd, x, y = egcd(a, b)

if x < 0:

x += b

return x

def encrypt(e, N, msg):

cipher = ""

for c in msg:

m = ord(c)

cipher += str(pow(m, e, N)) + " "

return cipher

def decrypt(d, N, cipher):

msg = ""

parts = cipher.split()

for part in parts:

if part:

c = int(part)

msg += chr(pow(c, d, N))

return msg

def main():

keysize = 32

e, d, N = generateKeys(keysize)

with open("rsa\_results.txt", "r") as msg\_file:

msg = msg\_file.read()

print(f"m:{msg}")

enc = encrypt(e, N, msg)

dec = decrypt(d, N, enc)

with open("rsa\_results.txt", "w") as file:

file.write(f"Message: {msg}\n")

file.write(f"e: {e}\n")

file.write(f"d: {d}\n")

file.write(f"N: {N}\n")

file.write(f"enc: {enc}\n")

file.write(f"dec: {dec}\n")

with open("rsa\_results.txt", "r") as file:

file\_contents = file.read()

print("\nResults from File:\n")

print(file\_contents)

main()