|  |  |  |
| --- | --- | --- |
|  |  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |
| **Институт комплексной безопасности и специального приборостроения**  Кафедра КБ-4 «Интеллектуальные системы информационной безопасности»  Дисциплина «Кроссплатформенная среда исполнения программного обеспечения» | | |
|  |  |  |

**Отчет**

по проектной работе на тему

«Реализицая алгоритма шифрования RSA на языке Python»

Выполнила:

Студент 2 курса

Группа ББСО-02-21

Шифр 21Б0613

Вандиловская Ксения Андреевна

Москва, 2022

1. **Задание проекта.**

Реализовать блочный шифр RSA на кроссплатформенном, интерпретируемом, языке программирования высокого уровня Python.

1. **Программный код с необходимыми комментариями и пояснениями.**

В функции main() происходит диалог с пользователем. Пользователь может согласиться использовать публичные и приватные ключи, выдаваемые программой либо ввести имя своего файла. Пользователь выбирает зашифровать или расшифровать сообщение. Потом вводит сообщение. Соответственно, верное введенное сообщение для расшифрования является последовательность из блоков.

Приватный ключ лежит в файле: private\_keys.txt

Публичный ключ лежит в файле: public\_keys.txt

Результат алгоритма лежит в файле: result.txt

Функция **chooseKeys()** - Выбирает два случайных простых числа из списка простых чисел, значения которых доходят до 100 тыс. Она создает текстовый файл и сохраняет два числа там, где они могут быть использованы позже. Используя простые числа, она также вычисляет и хранит открытый и закрытый ключи в двух отдельных файлах.

Функция **gcd(a, b)** - Выполняет алгоритм Евклида, возвращает НОД.

Функция **xgcd(a, b)** - Выполняет расширенный евклидов алгоритм, возвращает НОД, коэффициент a и коэффициент b.

Функция **chooseE(totient)** - Выбирает случайное число, 1 < e < totient, и проверяет, является ли оно взаимно простым с totient, то есть

gcd(e, totient) = 1

Функция **encrypt(message, file\_name=’public\_keys.txt’, block\_size=2)** - Шифрует сообщение (massege - string), повышая значение ASCII каждого символа до степени e и принимая модуль n. Возвращает строку чисел.

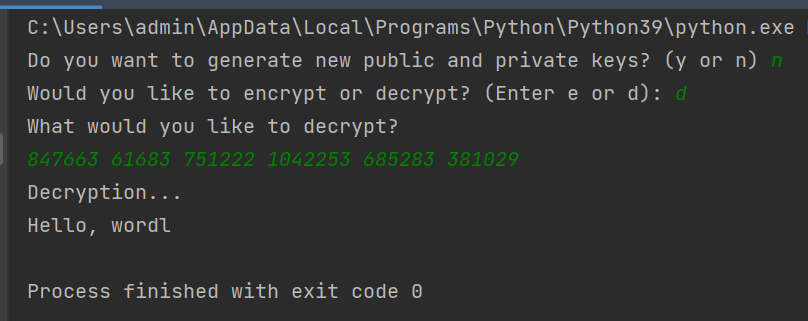
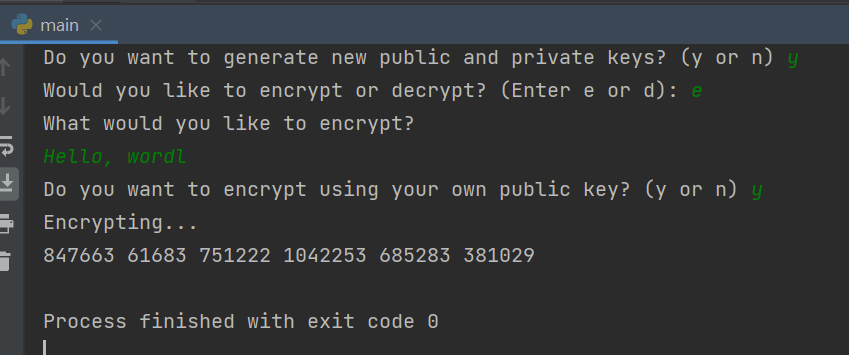
File\_name относится к файлу, в котором находится открытый ключ. Если файл не предоставлен, предполагается, что мы шифруем сообщение, используя наши собственные открытые ключи. В противном случае он может использовать чужой открытый ключ, который хранится в другом файле.

block\_size указывает, сколько символов составляет одну группу чисел в

каждом индексе encrypted\_blocks.

Функция **decrypt(blocks, block\_size=2)** - Расшифровывает строку чисел, возводя каждое число в степень d и принимая модуль n. Возвращает сообщение в виде строки. block\_size указывает, сколько символов составляет одну группу чисел в каждом индексе блоков.

**Пояснительная записка**

1. Реализация алгоритма шифрования RSA на Python.
2. Вандиловская Ксения, ББСО-02-21.
3. 
4. **Листинг кода.**

import random  
  
  
def gcd(a, b):  
  
 if (b == 0):  
 return a  
 else:  
 return gcd(b, a % b)  
  
  
def xgcd(a, b):  
  
 x, old\_x = 0, 1  
 y, old\_y = 1, 0  
  
 while (b != 0):  
 quotient = a // b  
 a, b = b, a - quotient \* b  
 old\_x, x = x, old\_x - quotient \* x  
 old\_y, y = y, old\_y - quotient \* y  
  
 return a, old\_x, old\_y  
  
  
def chooseE(totient):  
  
 while (True):  
 e = random.randrange(2, totient)  
  
 if (gcd(e, totient) == 1):  
 return e  
  
  
def chooseKeys():  
  
 rand1 = random.randint(100, 300)  
 rand2 = random.randint(100, 300)  
  
 fo = open('primes-to-100k.txt', 'r')  
 lines = fo.read().splitlines()  
 fo.close()  
  
 prime1 = int(lines[rand1])  
 prime2 = int(lines[rand2])  
  
 n = prime1 \* prime2  
 totient = (prime1 - 1) \* (prime2 - 1)  
 e = chooseE(totient)  
  
 gcd, x, y = xgcd(e, totient)  
  
 if (x < 0):  
 d = x + totient  
 else:  
 d = x  
  
 f\_public = open('public\_keys.txt', 'w')  
 f\_public.write(str(n) + '\n')  
 f\_public.write(str(e) + '\n')  
 f\_public.close()  
  
 f\_private = open('private\_keys.txt', 'w')  
 f\_private.write(str(n) + '\n')  
 f\_private.write(str(d) + '\n')  
 f\_private.close()  
  
  
def encrypt(message, file\_name='public\_keys.txt', block\_size=2):  
  
 try:  
 fo = open(file\_name, 'r')  
  
 except FileNotFoundError:  
 print('That file is not found.')  
 else:  
 n = int(fo.readline())  
 e = int(fo.readline())  
 fo.close()  
  
 encrypted\_blocks = []  
 ciphertext = -1  
  
 if (len(message) > 0):  
 ciphertext = ord(message[0])  
  
 for i in range(1, len(message)):  
  
 if (i % block\_size == 0):  
 encrypted\_blocks.append(ciphertext)  
 ciphertext = 0  
  
 ciphertext = ciphertext \* 1000 + ord(message[i])  
  
 encrypted\_blocks.append(ciphertext)  
  
 for i in range(len(encrypted\_blocks)):  
 encrypted\_blocks[i] = str((encrypted\_blocks[i] \*\* e) % n)  
  
 encrypted\_message = " ".join(encrypted\_blocks)  
  
 return encrypted\_message  
  
  
def decrypt(blocks, block\_size=2):  
  
 fo = open('private\_keys.txt', 'r')  
 n = int(fo.readline())  
 d = int(fo.readline())  
 fo.close()  
  
 list\_blocks = blocks.split(' ')  
 int\_blocks = []  
  
 for s in list\_blocks:  
 int\_blocks.append(int(s))  
  
 message = ""  
  
 for i in range(len(int\_blocks)):  
 int\_blocks[i] = (int\_blocks[i] \*\* d) % n  
  
 tmp = ""  
 for c in range(block\_size):  
 tmp = chr(int\_blocks[i] % 1000) + tmp  
 int\_blocks[i] //= 1000  
 message += tmp  
  
 return message  
  
  
def write\_res(s):  
 fo = open('result.txt', 'w')  
 fo.write(s)  
 fo.close()  
  
  
def main():  
 choose\_again = input('Do you want to generate new public and private keys? (y or n) ')  
 if (choose\_again == 'y'):  
 chooseKeys()  
  
 instruction = input('Would you like to encrypt or decrypt? (Enter e or d): ')  
 if (instruction == 'e'):  
 message = input('What would you like to encrypt?\n')  
 option = input('Do you want to encrypt using your own public key? (y or n) ')  
  
 if (option == 'y'):  
 print('Encrypting...')  
 s = encrypt(message)  
 print(s)  
 write\_res(s)  
 else:  
 file\_option = input('Enter the file name that stores the public key: ')  
 print('Encrypting...')  
 s = encrypt(message, file\_option)  
 print(s)  
 write\_res(s)  
  
 elif (instruction == 'd'):  
 message = input('What would you like to decrypt?\n')  
 print('Decryption...')  
 s = decrypt(message)  
 print(s)  
 write\_res(s)  
 else:  
 print('That is not a proper instruction.')  
  
  
main()