Cairo University

Faculty of Engineering
Cairo University

RSA Assignment

Course Name:

Cryptography

Supervised by:

Dr. Sameer
ENG. Khaled Moataz

Name: Asmaa Adel Abdelhamed Kawashty
Sec: 1
BN: 14
Code: 9202285

Overview:
I have implemented the RSA encryption algorithm using Python scripts(.py).
The project is divided into: common functions, chat module using sockets, RSA encryption and decryption, chosen ciphertext attack(mathematical model)

How to run:
I have used VS code, any IDE or code editor can be able to run python scripts.
I use Libraries: sockets, matplotlib, sympy.
So if you don't have these libraries, install them.

- pip install sockets
- pip install matplotlib
- pip install sympy

First, We should run the Sender, then the Receiver.
Both sender and receiver can send and receive messages, The sender sends to the receiver and vice versa time after time in order.

Explanation:

- **RSA Algorithm:**

I have implemented RSA through the following steps:

1. I generated the prime p, q with gererate_pq_primes function using a random prime number generator, and then get n, phi(n), e, d
2. Send the public key to the other person.
3. Read the message from the user.
4. Divide the message into sections, each section 5 characters, using splitToGroups function, and then convert the sections to integers using convertToInt function
5. encrypt each packet, and then send it to the receiver packet by packet.
6. The receiver receives each encrypted packet, then decrypts it, and then converts it back to character using convertToString function.

- **Utilities functions I used:**

- *GCD(a,b)*

Recursive function that takes numbers a, b and keeps trying to find the GCD of b and a%b

- *mod_inverse_solve(a,b)*

Uses pow(a, -1, b) function and generates d.

- *splitToGroups(message)*

Divide the message into groups, each group has 5 characters.

- *ConvertToInt(splited_message)*

Converts a string to an integer with Character Conversion as it’s explained in the assignment requirements

- *convertToString(number)*

Converts integer to a string by the inverse logic of ConvertToInt function.

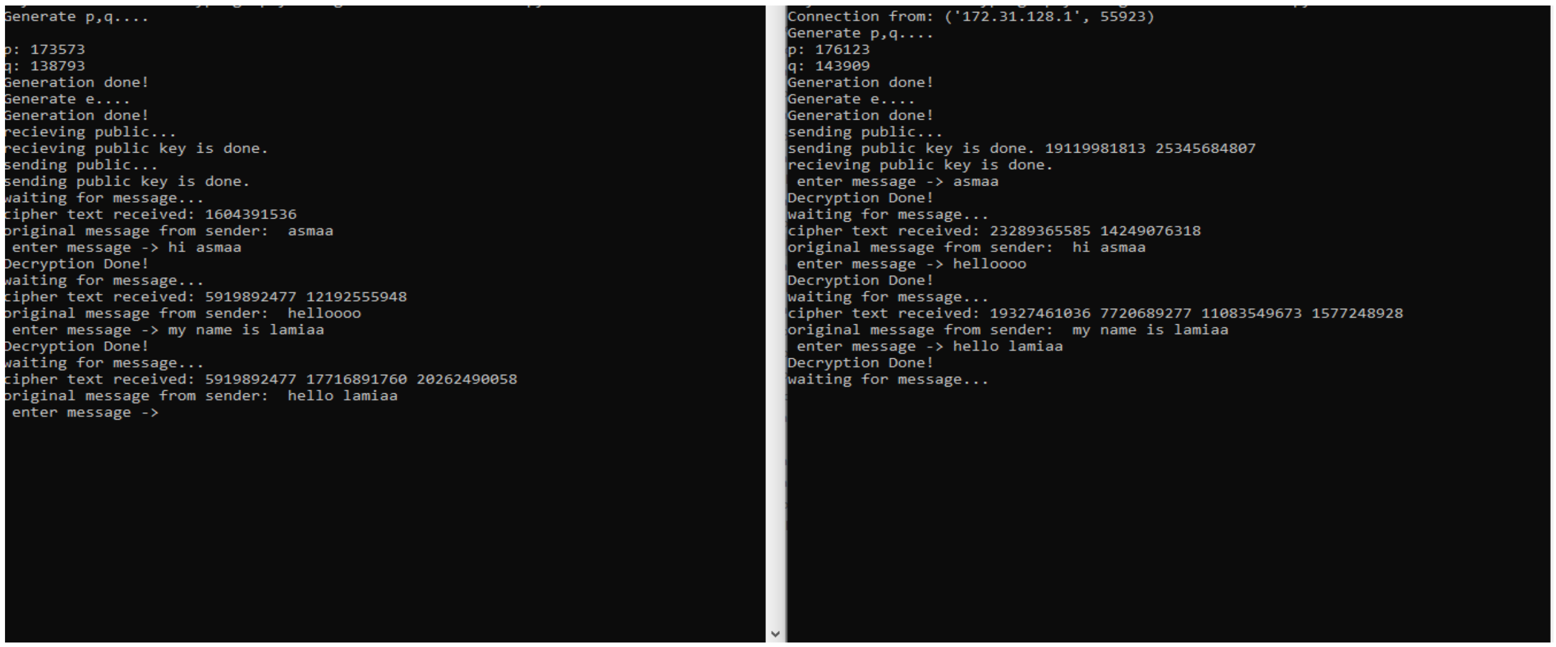
- *Encryption(message)*

Converts a message to an integer uses ConvertToInt function and then encrypt it by the public key(e, n)

- *Decryption(message)*

Decrypts the message using private key(d, n) and then Converts a it to a string uses convertToString function

- **Sender and Receiver:**



```
Generate p,q....

p: 173573
q: 138793
Generation done!
Generate e....
Generation done!
recieving public...
recieving public key is done.
sending public...
sending public key is done.
waiting for message...
cipher text received: 1604391536
original message from sender:  asmaa
 enter message -> hi asmaa
Decryption Done!
waiting for message...
cipher text received: 5919892477 12192555948
original message from sender:  helloooo
 enter message -> my name is lamiaa
Decryption Done!
waiting for message...
cipher text received: 5919892477 17716891760 20262490058
original message from sender:  hello lamiaa
 enter message ->
```

```
Connection from: ('172.31.128.1', 55923)
Generate p,q....
p: 176123
q: 143909
Generation done!
Generate e....
Generation done!
sending public...
sending public key is done. 19119981813 25345684807
recieving public key is done.
 enter message -> asmaa
Decryption Done!
waiting for message...
cipher text received: 23289365585 14249076318
original message from sender:  hi asmaa
 enter message -> helloooo
Decryption Done!
waiting for message...
cipher text received: 19327461036 7720689277 11083549673 1577248928
original message from sender:  my name is lamiaa
 enter message -> hello lamiaa
Decryption Done!
waiting for message...
```

First, the receiver generates p, q, e to be able to make the sender send messages to him and then he could decrypt it.

Then, he sends the public key to the sender, as soon as the sender receives the public key then he also generates p, q, e to be able to make the receiver send messages back to him then he could decrypt it, as the receiver he also sends the public key to receiver, then finally they start communication.

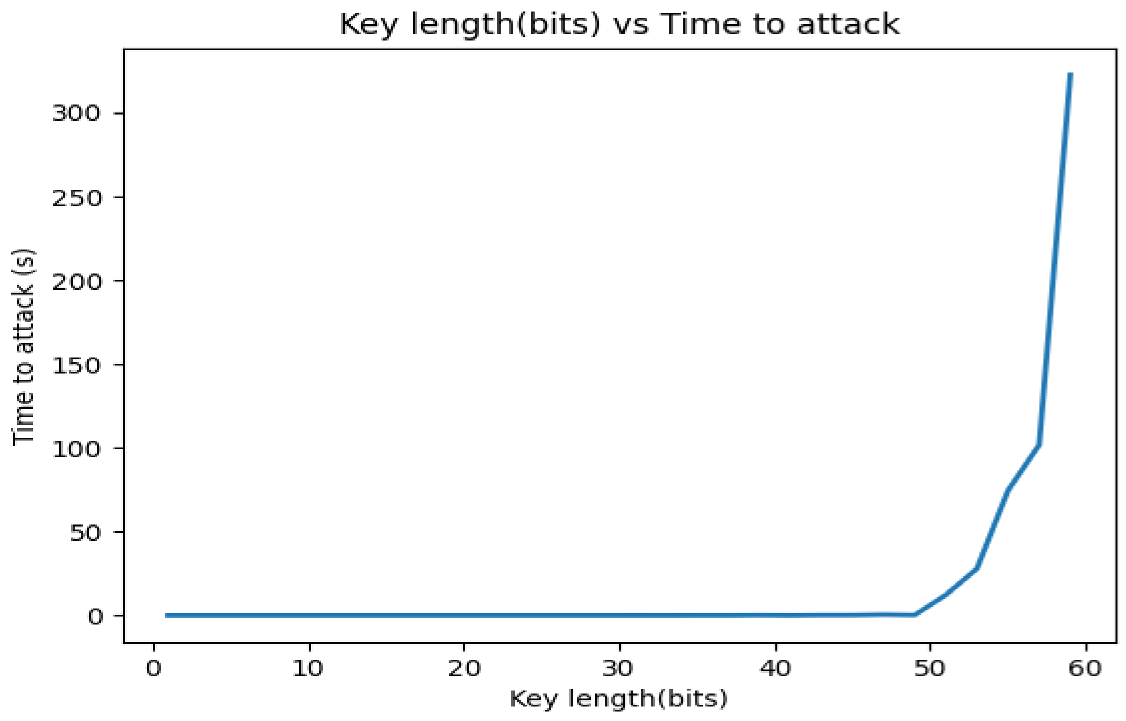
Each one of them sends a message once and then waits for the other to send a message back.

There is no limit of the message because we split it each block is 5 characters, and then, I convert it to integer with Character Conversion as it’s explained in the assignment requirements

- **Prime Factorization attack:**

I have implemented the attack by trying to divide n and get prime factorization for it(p, q). I use the “factorint” function from sympy.

The graph depicts attack against n up till 64 bits and graphs the time in seconds. It follows an exponential distribution, which is reasonable. Increasing the key length increases the time needed to try to break it exponentially. So The security of a cipher does not depend on the attacker not knowing the algorithm that was used for encryption. But it depends on how hard it is, mathematically, to break the code.



- **Encryption and Decryption time VS key Length:**

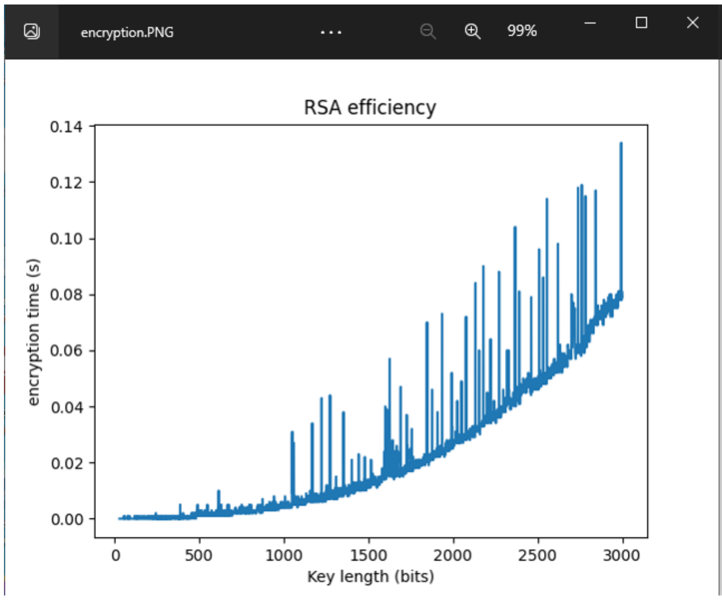
I start by reading p, q, e from a file that can be generated before computing the graph as it may take a lot of time.

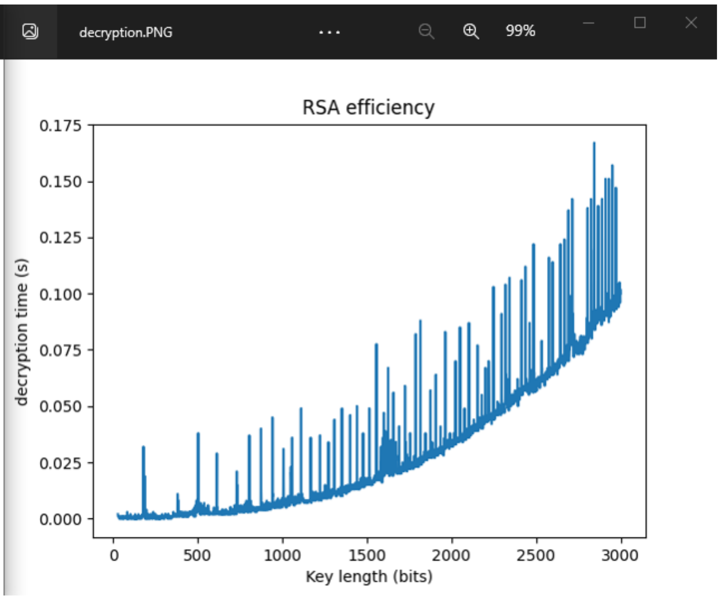
The graphing function tries each time to encrypt and decrypt the message and then calculate the time taken in both processing, store it in the list.

When I finish this process, then I plot graphs, one for encryption and the other of decryption.

The graph depicts length of key in binary bits against time taken for encryption, and decryption in seconds. The graph seems to be exponential which is logical.

I run from 27 bits to 3000 bits. The exponential isn't clear, because the numbers are very close to each other, and so small.





- **Code implementation:**

1. Encryption_functinons file is the class that has the implementation of encryption.
2. Decryption_functinons file is the class that has the implementation of decryption.
3. The sender, and receiver files are the files that create instances of the classes and then start the chat.

4. Efficiency file is the file that tests the encryption and decryption time vs key length.
5. Attack file is the file that tests the attack, and attack time, when you run it and choose 1 then you will test the attack, 2 then you will test the time vs attack.
6. Common_functions file is the file that has all common functions that are used in different files.
7. Generate_key is the file that has the implementation of generating e

Note:

- In each place that I want to use (%) to calculate mod, I use pow(a, b, c) instead because it's more efficient, the result will be a**b mod c.
- After implementing some functions, I found the same functions in libraries like sympy, and math. The difference is that their functions are so fast, so I commented on my functions and used their functions.
- I commented the lines where I generate p, q, in efficiency.py, and attack.py, because they take a lot of time in case anyone wants to run quickly, but if anyone wants to test the functions from scratch, uncomment them.