

Implement the RSA algorithm

Encryption:

- 1) Convert the message to integer to can apply mathematical operation on in (each char convert to 8-bit unique number).
- 2) Use public key to encrypt the message to get the cipher $C = M^e \bmod n$.
- 3) Convert the cipher to string to transiminate it through the socket channel.

Decryption:

- 1) Convert the Cipher string to integer to can apply mathematical operation on in (reverse the last step in encryption).
- 2) Use private key to decrypt the cipher to get the Message $M = C^d \bmod n$
- 3) Convert the Message number to string to can be readable

Generate public key

- 1) Read P and Q from external file to can be change
- 2) Calculate $\phi(n) = (P-1) * (Q-1)$
- 3) Calculate $n = P * Q$
- 4) Generate random e which satisfy the condition $\text{GCD}(e, \phi(n)) = 1$
- 5) Return (e, n)

Generate Private key

- 1) Calculate $n = P * Q$
- 2) Calculate $\phi(n) = (P-1) * (Q-1)$
- 3) Calculate d using e and $\phi(n)$ call the invertModule $e * d = 1 \pmod{\phi(n)}$
- 4) Return (d, n)

All this functions are implemented in RSA.py file and will be used later

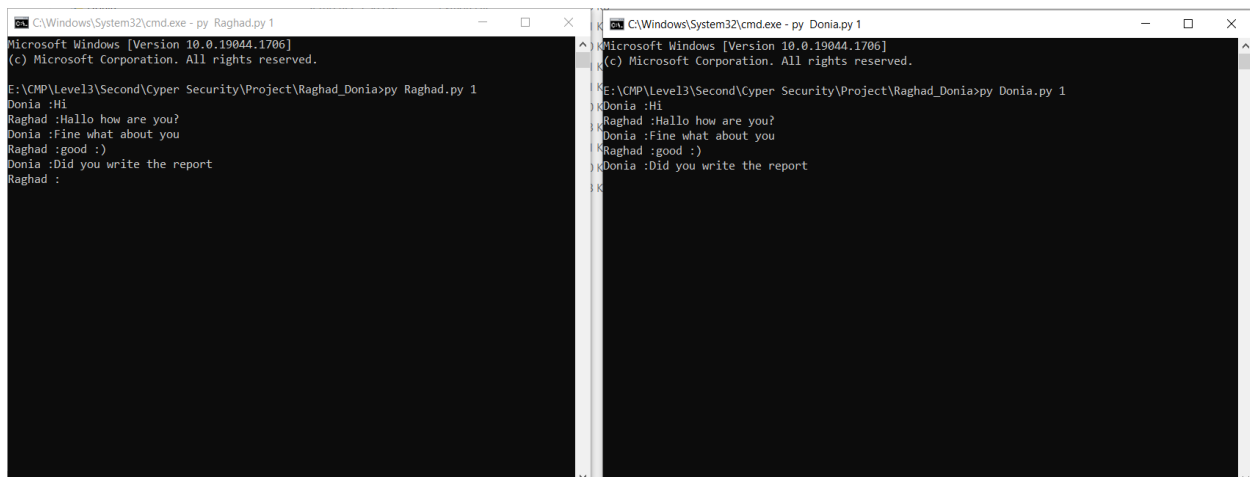
The project has mainly tow files Raghad.py and Donia.py. Both files can send and Receive

Raghad.py read the P and Q from PQ.txt, then generate the public and private key and send the public key to Donia.py to use in encryption and keep the private key to can be used in decryption.

Donia.py read the P and Q from PQ2.txt, then generate the public and private key and send the public key to Raghad.py to use in encryption and keep the private key to can be used in decryption.

The files can work in one of two modes which send as an argument when run the script

Frist mode is the chat both files get the message from the terminal



```
C:\Windows\System32\cmd.exe - py Raghad.py 1
Microsoft Windows [Version 10.0.19044.1706]
(c) Microsoft Corporation. All rights reserved.

E:\CMP\Level3\Second\Cyper Security\Project\Raghad_Donia>py Raghad.py 1
Donia :Hi
Raghad :Hallo how are you?
Donia :Fine what about you
Raghad :good :)
Donia :Did you write the report
Raghad :
```

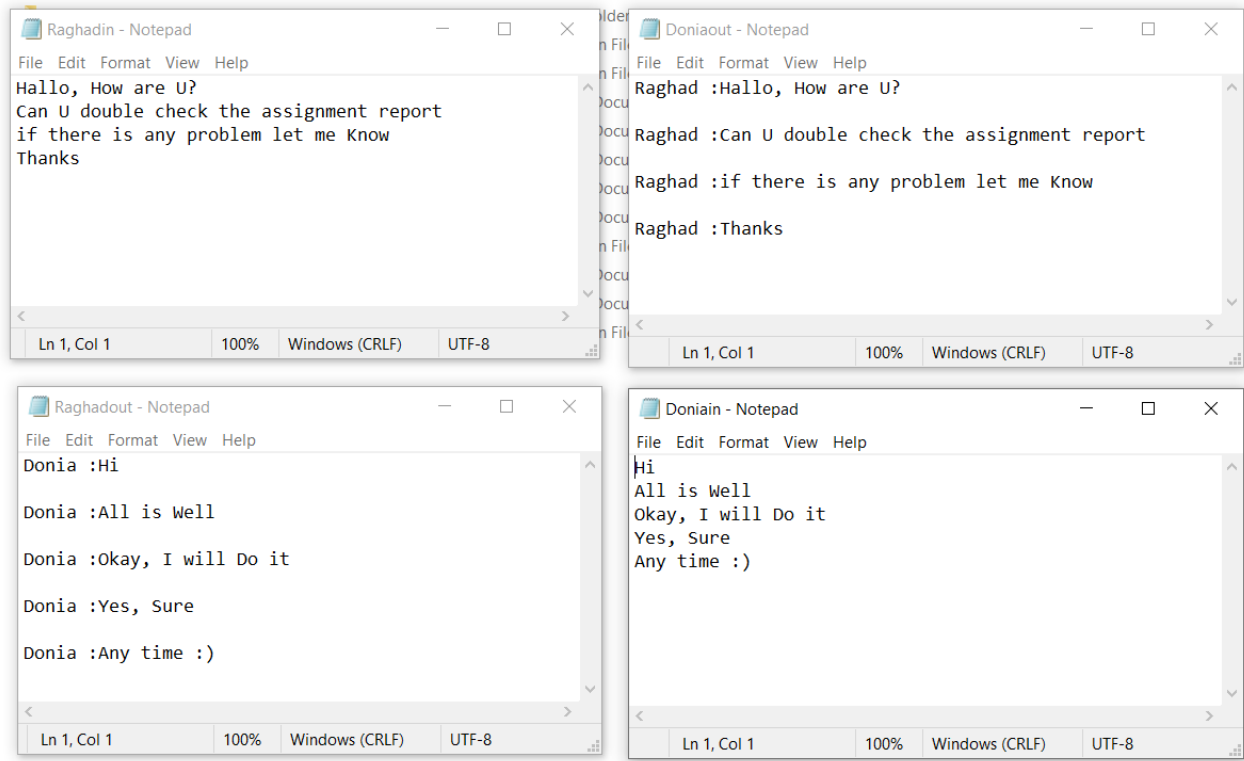
```
C:\Windows\System32\cmd.exe - py Donia.py 1
Microsoft Windows [Version 10.0.19044.1706]
(c) Microsoft Corporation. All rights reserved.

E:\CMP\Level3\Second\Cyper Security\Project\Raghad_Donia>py Donia.py 1
Donia :Hi
Raghad :Hallo how are you?
Donia :Fine what about you
Raghad :good :)
Donia :Did you write the report
Raghad :
```

Second mode is read from Sample input files

Raghad.py read messages from Raghadin.txt and encrypt it to send to Donia.py
And the receive messages from Donia.py after decryption is saved in Raghadout.txt which should be the sampled output file

Donia.py read messages from Doniain.txt and encrypt it to send to Raghad.py
And the receive messages from Raghad.py after decryption is saved in Doniaout.txt which should be the sampled output file



Mathematical Brute Force attack for RSA

It's based on the idea of factorization, as p , q are primes, then the only factors of n are p , q

- 1) Iterate from $i = 3$ up to \sqrt{n} with step size = 2
- 2) Each iteration, try to divide n by i , if the remainder = 0, then $i = p$
- 3) Thus, $q = n / p$

For Testing:

- 4) Read n & the cipher from the text file
- 5) Apply the MA, to get p , q
- 6) Generate private keys d , n
- 7) decrypt the cipher and compare it with its plain text

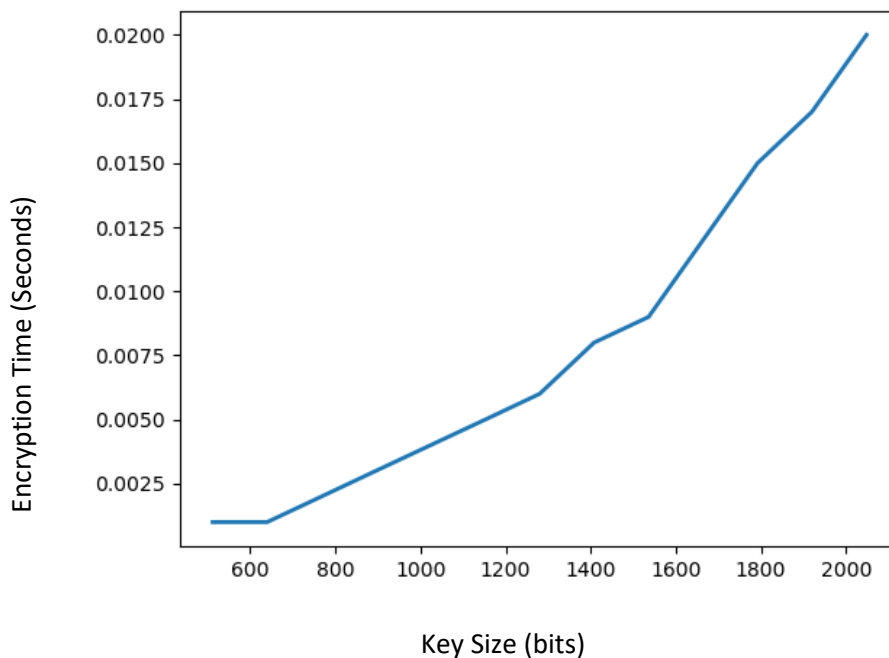
Chosen Cipher Text attack for RSA

When run the two files of Raghad.py and Donia.py the cipher Message generated by Donia.py and the public key of ragahd.py are store in publicData.txt

After that can run CCA.py file which

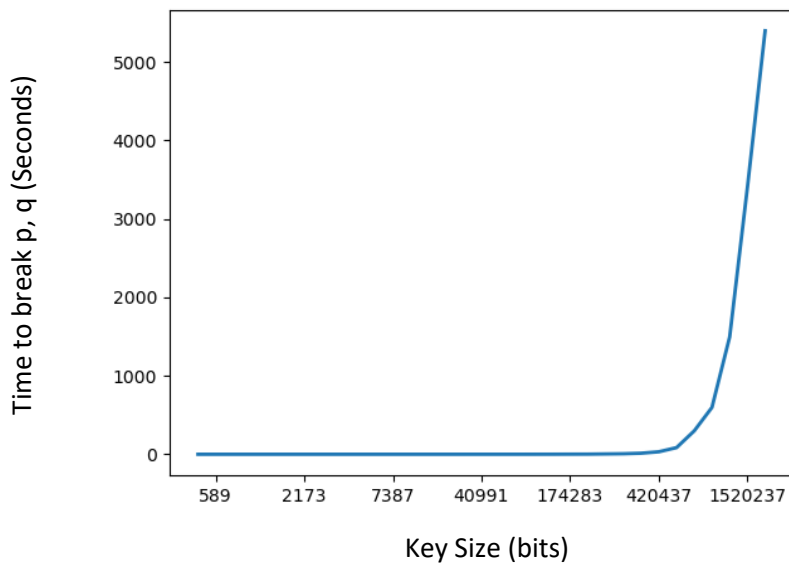
- 8) red the public key of Raghad.py (n,e)
- 9) red the cipher message one by one
- 10) generate random r with condition $\text{GCD}(n,r)=1$
- 11) calculate $C' = C * r^e \text{ mod } n$
- 12) send C' to Raghad.py after convert to string
- 13) Raghad.py run in attack mode (Mode 3) decrypt the message C'
- 14) After decryption ($Y = (C')^d \text{ mod } n$)resend it to CCA.py
- 15) Calculate r^{-1} using r and n using the invertModule $r * r^{-1} = 1 \text{ (mod } n)$
- 16) Calculate the message $M = r^{-1} * Y \text{ mod } n$
- 17) Show the result of the attack in the terminal

RSA Encryption Efficiency



For Large n (>256 bits), the encryption will be slower, and this make sense, because the mod operations takes long time with big integers

MA Efficiency



As shown in the figure, it takes very short time to attack a small key, but in large keys (>64 bits), the time to break the key is huge, and that's why RSA uses large p, q

Appendix

For plotting, I use a function to generate random primes with n bit size:

```
def generate_big_prime(n):
    found_prime = False
    while not found_prime:
        p = randint(2**(n-1), 2**n-1)
        if is_prime_optimized(p, tests_count):
            return p
```

It generates random integer that has n bits and loop until the generated random is prime

For checking the primality of a number, we use the **Fermat's Little Theorem**,

With test count = 10000

```
def is_prime_optimized(n, testsNumber):
    ## applying fermat's primality test
    if n == 1:
        return False
    if testsNumber >= n:
        testsNumber = n - 1
    for x in range(testsNumber):
        rand = randint(1, n - 1)
        if pow(rand, n-1, n) != 1:
            return False
    return True
```