COMPUTER SCIENCE INVESTIGATORY PROJECT

2020-2021

RSA CRYPTOGRAPHY DEMONSTRATION GUI

DECLARATION

I hereby declare that this project report has been originally carried under the guidance and supervision of Ms. Anuja Ghosh, Computer Science teacher of CMR National Public School, Bengaluru, Karnataka.

Aathira S

ACKNOWLEDGMENT

I wish to express my deep gratitude and sincere thanks to the Head of the Institution, CMR National Public School, Bengaluru, Karnataka for her encouragement and for all the facilities she provided for the project work. I extend my hearty thanks to Anuja Ma'am, Computer teacher, who guided me to the successful completion of this project. I take this opportunity to express my deep sense of gratitude for her invaluable guidance, constant encouragement, constructive comments and motivation, which has sustained my efforts at all stages of this project work.

I can't forget to offer my sincere thanks to my family who helped to carry out this project work and their valuable advice and support.

Aathira S

INDEX

INTRODUCTION

In today's world where our information isn't safe anymore as it has gone 'online', we have created a program depicting a modern system where all the messages and information is encrypted and decrypted using a 'key'. Our project shows an example of RSA cryptology which uses this concept of keys.
We have used an asymmetric key type,the RSA(Rivest - Shamir - Adleman) method of cryptography. This project can encrypt the normal text to cipher text and decrypt the cipher text to normal. Our code is python tkinter gui based, which allows an interface for the RSA demo.

The concepts used in this project:

1) User-Defined Modules
2) User-Defined Functions
3) Tkinter Module
4) Random Module

ABOUT OUR PROJECT

MAIN PURPOSE

The main purpose of our project is to provide security for transfer of sensitive information using RSA cryptology through python in a simplified manner.

PROGRAMMING LANGUAGE USED FOR DEVELOPMENT

The programming language we used to show the functioning of this project is python.

Python is currently the most widely used multi-purpose, high-level programming language. It allows programming in Object-Oriented and Procedural patterns. Python programs generally are smaller than other programming languages like Java. Programmers have to type relatively less and the indentation requirement of the language makes them readable all the time.

Python language is being used by almost all tech-giant companies like – Google, Amazon, Facebook, Instagram, Dropbox, Uber… etc.

The biggest strength of Python is huge collection of standard library which can be used for the following:

- Machine Learning
- GUI Applications (like Kivy, Tkinter, PyQt etc.)
- Web frameworks like Django (used by YouTube, Instagram, Dropbox)
- Image processing (like OpenCV, Pillow)
- Web scraping
- Testing frameworks

FEATURES

Imagine you have some important document that you don’t want others to see. You can put a lock-password on it so that no one can enter without a password. But that can be easily cracked. The next, most ideal step will be to change the contents of your document (encrypting) in such a manner that it looks very strange at the first look. But there will be a pattern behind it, for which upon decoding you can view and understand the contents of the file.

The main features of our project are:

- The program will encrypt a file so that to another user (potential spy) it appears in a non-readable format which they cannot understand.
- The encryption happens using the concept of ‘keys’. The keys are used to encrypt-decrypt the contents of a file. Each user has a key pair. If you encrypt a file with one key, you can decrypt only using the key pair. Since the key pairs are long, they cannot be memorized easily.

The mode of cryptology we use provides a huge amount of security and our program utilizes and presents this concept in a very simple code/format.

CRYPTOGRAPHY

INTRODUCTION

Cryptography is a method of protecting information and communications through the use of codes, so that only those for whom the information is intended can read and process it. The prefix “crypt-” means “hidden” or “vault”, and the suffix ”-graphy” stands for “writing”. In computer science, cryptography refers to secure information and communication techniques derived from mathematical concepts and a set of rule-based calculations called algorithms, to transform messages in ways that are hard to decipher.

HISTORY

Before the modern era, cryptography focused on message confidentiality (i.e., encryption)—conversion of messages from a comprehensible form into an incomprehensible one and back again at the other end, rendering it unreadable by interceptors or eavesdroppers without secret knowledge (namely the key needed for decryption of that message).

- Encryption attempted to ensure secrecy in communications, such as those of spies, military leaders, and diplomats.
- In recent decades, the field has expanded beyond confidentiality concerns to include techniques for message integrity checking, sender/receiver identity authentication, digital signatures, interactive proofs and secure computation, among others.

TYPES

<u>Symmetric Key Cryptography</u>
It is an encryption system where the sender and receiver of the message use a single common key to encrypt and decrypt messages. Symmetric Key Systems are faster and simpler, but the problem is that the exchange of the key should happen in a secure manner.

<u>Asymmetric key cryptography</u>
In this project we have used the concept of Asymmetric Key Cryptography.

Under this system a pair of keys is used to encrypt and decrypt information. A public key is used for encryption and a private key is used for decryption. Public key and Private Key are different. Even if the public key is known by everyone the intended receiver can only decode it because he alone knows the private key.

CRYPTANALYSIS AND SECURITY

Cryptanalysis is the process of studying cryptographic systems to look for weaknesses or leaks of information. It is used to breach cryptographic security systems and gain access to the contents of encrypted messages, even if the cryptographic key is unknown. Using this feature, we can improvise and strengthen a systems security. It can provide

- Data Integrity - Data integrity means assurance and maintenance of data accuracy and consistency. It can apply to a stream of messages, a single message, or selected fields within a message. A loss of integrity is the unauthorized modification or destruction of data.

- Data Confidentiality - Preserving authorized restrictions on information access and disclosure, including means for protecting personal privacy and proprietary information. A loss of confidentiality is the unauthorized disclosure of information.

- Authenticity
  Provide authentication to all the node and base stations for utilizing the available limited resources. It also ensures that only the authorized node can participate in the communication.

- Nonrepudiation
  Nonrepudiation prevents either sender or receiver from denying a transmitted message. Thus, when a message is sent, the receiver can prove that the alleged sender in fact sent the message. Similarly, when a message is received, the sender can prove that the alleged receiver in fact received the message.

- Access Control
  Access control is the ability to limit and control the access to host systems and applications via communications links. To achieve this, each

entity trying to gain access must first be identified, or authenticated, so that access rights can be customized to the individual.

PUBLIC KEY CRYPTOGRAPHY

INTRODUCTION

Public key encryption is also known as asymmetric or public key cryptography. It is a method of encrypting data with two different mathematically related keys and making one of the keys, the public key, available for anyone to use. The other key is known as the private key. Data encrypted with the public key can only be decrypted with the private key, and data encrypted with the private key can only be decrypted with the public key.

Symmetric cryptography was well suited for organizations such as governments, military, and big financial corporations which were involved in the classified communication.
With the spread of less secure computer networks in the last few decades, a genuine need was felt to use cryptography at a larger scale. The symmetric key was found to be non-practical due to challenges it faced for key management. This gave rise to the public key cryptosystems.

WORKING

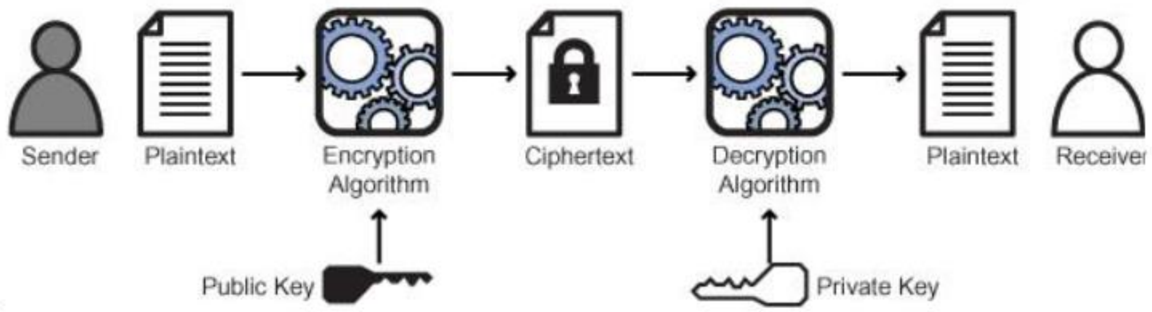
<u>Components of public key cryptosystems</u>

- **Plaintext**: the readable message or data that is fed into the algorithm as input.
- **Encryption algorithm**: It performs various transformations on the plaintext to produce the ciphertext.
- **Public Key and Private Key**: This is a pair of keys that have been selected so that one is used for encryption and the other is used for decryption. The exact transformations performed by the algorithm depend on the public or private key that is provided as input.
- **Ciphertext**: This is the scrambled message produced as output. It depends on the plaintext and the key. For a given message, two different keys will produce two different ciphertexts.
- **Decryption algorithm**: This algorithm accepts the ciphertext and the matching key and produces the original plaintext.

Process of sending and receiving message

1. Each user generates keys to be used for the encryption and decryption of messages.
2. Each user places one of the two keys in a public register or other accessible file. This is the public key. The companion key Is kept private. Each user maintains a collection of public keys obtained from other users.
3. If Bob (sender) wishes to send a confidential message to Alice (receiver), Bob encrypt the message using Alice's public key.
4. When Alice receives the message, she decrypts it using her private key. No other recipient can decrypt the message because only Alice knows Alice's private key.

The process of encryption and decryption is depicted in the following illustration −



Requirements of Public Key Cryptosystems

Diffie and Hellman postulated a Public Key Cryptographic system and laid out the following conditions that such algorithms must fulfil.
1. It is computationally easy for a party B to generate a key pair (public key PU_b and private key PR_b)
2. It is computationally easy for a sender A, knowing the public key and message to be encrypted, M, to generate the corresponding ciphertext C

   $C = E(PU_b , M)$
3. It is computationally easy for receiver B to decrypt the resulting ciphertext using the private key to recover the message:

   $M = D(PR_b , C)$

4. It is computationally infeasible for an adversary, knowing the public key PU_b to determine the private key PR_b

   $Y = f(X)$ easy
5. It is computationally infeasible for an adversary, knowing the public key, Pub, and a ciphertext, C to recover the original message, M.

   $X = f^{-1}(Y)$

APPLICATIONS

In broad terms, we can classify the use of public key cryptosystems into the following 3 categories:

- **Encryption/Decryption:** The sender encrypts a message using the recipient's public key
- **Digital Signature:** The sender 'signs' a message with its private key. Signing is achieved by a cryptographic algorithm applied to the message or a small block of data that is a function of the message.
- **Key exchange:** Two sides cooperate to exchange a session key. Several different approaches are possible, involving the private key(s) of one or both parties.

NOTE: Some assurance of the authenticity of a public key is needed in this scheme to avoid spoofing by adversary as the receiver. Generally, this type of cryptosystem involves a trusted third party which certifies that a particular public key belongs to a specific person or entity only.

COMPARISON BETWEEN CONVENTIONAL AND ASYMMETRIC CRYPTOGRAPHY

Below we discuss some differences regarding their requirements.

Conventional Encryption	Public-Key Encryption
Needed to work:	
The same algorithm with the same key is used for encryption and decryption.	One algorithm is used for encryption and a related algorithm for decryption with a pair of keys, one for encryption and one for decryption.

The sender and receiver must share the algorithm and the key.	The sender and receiver must each have one of the matched keys (not the same one).
Needed for security:	
The key must be kept secret.	One of the two keys must be kept secret.
It must be impossible or atleast impractical to decipher a message if the key is kept secret.	It must be impossible or atleast impractical to decipher a message if one of the keys is kept secret.
Knowledge of the algorithm plus samples of cipher text must be insufficient to determine the key	Knowledge of the algorithm plus one of the keys must be insufficient to determine the other key.

Below we discuss differences regarding their key characteristics.

Differentiator	**Symmetric Key Encryption**	**Asymmetric Key Encryption**
Symmetric Key vs Asymmetric key	Only one key (symmetric key) is used, and the same key is used to encrypt and decrypt the message.	Two different cryptographic keys (asymmetric keys), called the public and the private keys, are used for encryption and decryption.
Complexity and Speed of Execution	It's a simple technique, and because of this, the encryption process can be carried out quickly.	It's a much more complicated process than symmetric key encryption, and the process is slower.
Length of Keys	The length of the keys used is typically 128 or 256 bits, based on the security requirement.	The length of the keys is much larger, e.g., the recommended RSA key size is 2048 bits or higher.

Usage	It’s mostly used when large chunks of data need to be transferred.	It’s used in smaller transactions, primarily to authenticate and establish a secure communication channel prior to the actual data transfer.
Security	The secret key is shared. Consequently, the risk of compromise is higher.	The private key is not shared, and the overall process is more secure as compared to symmetric encryption.
Ease of Distribution	Difficult as the shared key must be kept secret among the multiple people to whom it is known.	Easy as the public key need not be kept secret from anyone and the private key is to be known only to its owner.
Examples of Algorithms	Examples include RC4, AES, DES, 3DES, etc.	Examples include RSA, Diffie-Hellman, ECC, etc.

While using asymmetric cryptography we face various problems such as
- Repudiation
- Differentiating between two parties who share a key
- Trust and safety of ahring keys
- Multiple places to protect the key

Such problems can be solved by adopting asymmetric cryptosystems

Hybrid cryptosystems employ the advantages of both systems to provide better solutions for modern file transfer systems. For example, a secret file (especially a large one) will be encrypted by “symmetric cryptosystem” while using “asymmetric cryptosystem” to encrypt the symmetric key.

RSA ALGORITHM

INTRODUCTION

RSA (Rivest - Shamir - Adleman) is one of the oldest public-key cryptosystems used worldwide for the secure transmission of data. Named after its designers Ron Rivest, Adi Shamir, and Leonard Adleman, it was published in 1977. Being an asymmetric cryptography algorithm, it works on the concept of a publicly accessible public key and a secretly kept private key such that anyone could encode a message with the public key but only one with the private key can decode it to the original.
This algorithm is used in the web browsers, email, VPNs, chat and other communication channels we use today and extensively used in Bluetooth, MasterCard, VISA, e-banking, e-communication, e-commerce platforms alike for the extreme secure communication it ensures.

OBJECTIVES OF RSA

Using the Diffie-Hellman concept, the founders had perfectly formulated RSA encryption methods used today.
One of the main goals of the trio founders was to work out a one-way, irreversible function such that the only way to decode the message was with a special piece of information not known to the rest of the world, that is the *private key*.
The key pair also involves a common number, starting from a minimum of 1024 bits in practical usage, that is a product of 2 primes, whose working we'll delve into later parts of the report. This makes it extremely hard to tap into the security of this algorithm because knowing the encryption method as well as 1 key of the pair, it is practically infeasible to crack the other due to the hard nature of *factorization* of such a large number.
Thus, these aspects ensure the authenticity and confidentiality that RSA was worked upon to achieve.

METHOD OF ENCRYPTION AND DECRYPTION

We encrypt our message M using the public encryption key that is of the form:

- (e,n) ⇒ where e and n are positive integers

- Firstly, the message M is converted into a numeric equivalent of an integer that lies between 0 and n-1 ⇒ **0 <= M <= n-1**
  In the case of a long message, it is broken into blocks, each of which is represented as an integer.

- To encrypt the message M is raised to power e followed by modulo n, to get the ciphertext C.
  Here, we represent encryption and decryption of a message and ciphertext respectively as E(M) and D(C) :
  $$C = E(M) = M^e \pmod n \Rightarrow \text{ for message } M$$
  The decryption of ciphertext follows the same function but using the private key of the form :
  - (d,n) ⇒ d,n being positive integers again
    $$M = D(C) = C^d \pmod n \Rightarrow \text{ for ciphertext } C$$

→ How is the appropriate key pair chosen for the RSA method?

The number n common in the 2 keys, is the product of 2 very large and random primes p and q

$$n = p * q$$

p and q being such large numbers make sure that it is impossible to factorize the number n (which is typically at least 512 bits in modern day application), that is public, and to retrieve the original factors p and q from it.

- We pick a number e which is co-prime to the number,
  ϕ (n) = (n-1). (q-1) (where ϕ(n) is the Euler totient function)

  $$\gcd(e , \phi) = 1 \Rightarrow \text{gcd stands for greatest common divisor}$$

- The integer d is calculated using the following relation and it turns out to be multiplicative inverse of e, mod ϕ(n)

$$e.d \mod \phi(n) \equiv 1$$

OR

$$e.d \equiv 1 \pmod{\phi(n)}$$

- Hence we obtain the key pair of the public encryption key (e,n) and private decryption key (d,n)
  To summarise the variables and their properties :

p , q → 2 large prime numbers	Private and chosen
n = p.q	Public and calculated
e (using gcd(e , ϕ(n)) = 1)	Public and chosen
d ($d \equiv e^{-1} \bmod \phi(n)$)	Private and calculated

Underlying Mathematics

For any integer M (here, message) that is coprime with n ,

$$M^{\phi(n)} \equiv 1 \pmod{n} \Rightarrow (1)$$

ϕ(n) stands for the Euler totient function which gives the number of positive integers less than n that are co-prime with n :

- For primes (like p and q) , ϕ(p) = p-1
- For n = p.q as said before,

$$\begin{aligned}\phi(n) &= \phi(p).\phi(q)\\ &= (p-1).(q-1)\\ &= n - (p+q) +1\end{aligned}$$

→ e is co-prime with ϕ(n), and multiplicative inverse d is calculated such that

$$e.d \equiv 1 \pmod{\phi(n)}$$

→ If e and d are chosen correctly even the decryption works correctly,

$$D(E(M)) \equiv (E(M))^d \equiv (M^e)^d \pmod{n} \equiv M^{e.d} \pmod{n}$$
$$E(D(M)) \equiv (D(M))^e \equiv (M^d)^e \pmod{n} \equiv M^{e.d} \pmod{n}$$

→ $M^{e.d} \equiv M^{k\,\phi(n)+1}$ (for some integer k)

→ Using (1) for an M that is not divisible by p

$M^{p-1} \equiv 1 \pmod{p}$

(p-1) clearly divides ϕ(n) and using modulus properties,

$$M^{k\,\phi(n)+1} \equiv M \pmod{p}$$

Similarly for q also,

$$M^{k\,\phi(n)+1} \equiv M \pmod q$$

→ Combining the 2 equations for p and q,

$$\mathbf{M^{e.d} \equiv M^{k\,\phi(n)+1} \equiv M \pmod n}$$

This implies that **E(D(M)) = M** and **D(E(M)) = M** for all M where $0 \le M < n$.

ILLUSTRATION

Consider the simple case of

- p=47 q=59 n=p.q=2773
- We take e=157
- phi=46.58=2668
- Calculating d using the algorithm we get,
  d=17

Taking the letters as number equivalents of its rank in the english alphabet such as A=00, B=01, C=02 … till Z=26

→ Let the message be 'HI THERE'

→ We take the whole message as blocks of 2 letters each as below :

HI TH ER E

→ Whose number equivalent is,

0809 2008 0518 0500

→ Our value of e is 10001 in binary , we encode the 1st block with M=809

$M^{17} = (((((1)^2 \,.\, M)^2)^2\,)^2)^2 \,.M = 1252 \pmod{2773}$

→ This way the whole message is encrypted as ,

1252 0683 1364 0728

SECURITY PROVIDED BY RSA

RSA protects the system from many threats. Keeping in mind of the following situations, the text is encrypted accordingly

1. Plain text attacks:
It is classified into 3 subcategories: -

- Short message attack:
  In this we assume that the attacker knows some blocks of plain text and tries to decode cipher text with the help of that. So, to prevent this pad (Add unnecessary text at start and end of encryption) the plain text before encrypting.
- Cycling attack:
  This attacker will think that plain text is converted into cipher text using permutation and he will apply right for conversion. But the attacker does not write plain text. Hence will keep doing it.
- Unconcealed Message attack:
  Sometimes it happens that plain text is the same as cipher text after encryption. So, it must be checked that it cannot be attacked.

2. Chosen cipher attack:
This attacker is able to find out plain text based on cipher text using Extended Euclidean Algorithm.

3. Factorization attack:
If an attacker is able to know P and Q using N, then he could find out the value of the private key. This can be failed when N contains at least 300 longer digits in decimal terms, which the attacker will not be able to find. Hence it fails.

4. Attacks on Decryption key:

- Revealed decryption exponent attack:
  If the attacker somehow guesses decryption key D, not only the cipher text generated by encryption the plain text with corresponding encryption key is in danger, but even future messages are also in danger. So, it is advised to take fresh values of two prime numbers (i.e., P and Q), N and E.

DRAWBACKS OF RSA ALGORITHM

1. RSA is a public key cryptosystem (asymmetric cryptography) which is slow compared to symmetric cryptography.

2. It requires a more computer power supply compared to single key encryption.

3. In this cryptosystem, if the private key is lost then all received messages cannot be decrypted but security wise, it's great.
4. Complexity of algorithm i.e., key is too large and calculation time is long.
5. Very slow key generation.

SYSTEM REQUIREMENTS

The following is a comprehensive list of the minimum requirements to be able to execute this project successfully

- Operating system
- Python 3
- Source code
- Hard disk space – 30 KB
- Memory – 30 MB
- Python’s tkinter library
- Python’s random module
- Computer peripherals such as keyboard, mouse, monitor, etc.

FUNCTIONS AND CLASSES

USER-DEFINED FUNCTIONS AND CLASSES USED IN OUR PROGRAM

Name	Return Type	Description
	encrypt/decrypt module	
encdec(mblock,e,n)	list	• Inputs a list of integers as mblock (message block) • Encrypts/Decrypts each value in mblock (message block) • Returns a list of integers called C corresponding to their respective decryptions/encryptions makeblock
makeblock(m,n)	list	Breaks entire string message m into substrings of size based on length of n, and stores into list msgblock after converting to number for each substring using ascii number
breakblock	str	Converts each element of list (lis) into the corresponding substring it represents using ascii value and concatenates the whole to reproduce the full message string (msg)
encrypt	list	Encrypts the message string m using the key k and encdec and makeblock function(user-defined) into cipher text list content
decrypt	str	Decrypts the cipher text list c using the key k and encdec and breakblock functions (user- defined) into final message string (final_msg)
	primenos.py creates file primenos.txt and inserts 1-4 digit prime prime numbers are generated using sieve method	
gcd(a,b)	Int	Performs long division method to find gcd of integers a,b

euclidgcd(a,b)	Int	Performs the extended Euclidean algorithm Returns the gcd, coefficient of a, and coefficient of b in eqn gcd(a,b) = ax + by
getpq()	Int	opens file primenos.txt and choose two distinct numbers p,q from it and returns them
main(p=0,q=0)	Int	• generates RSA key pairs • may take input for primes p,q or chooses random primes • computes RSA key pair using these primes • returns RSA key pairs (e,n) and (d,n) as values of e,d,n
RSA DEMO		
class user: __init__(self,name,colour,pd)		
checkpq()	None	Check validity of p and q for key generation and perform operations accordingly
newkeypair(self,p=0,q=0)	None	Creates new key pair and assigns to self
createbal(widget,msg)	None	Creates a balloon message (alternate text) for the widgets
helpbtn()	None	Creates a help button
write(self)	None	Lets you write a message
getinfo()	None	Encrypts the typed message
read(self)	None	Lets you read the received message
decndis()	None	Decrypts the received message
checklog()	None	Checks if password/username is correct

ALGORITHM

1) ENCRYPTION AND DECRYPTION

Encryption and decryption follows the same method with the difference that we use a different key value for the process

1. Message **M** is broken into a list of integer blocks
   i. [M_0, M_1... M_x]

2. Start traversal from message blocks list from index 0 to x and repeat steps a to d for all. For message block M_j

   a. binary number e is of the form e_k e_{k-1} ... e_1e_0
   b. Set variable c=1
   c. Start traversal through e from i=k to 0 and repeat the following steps (i) and (ii)
      (i) Set c to remainder of c^2 when divided by n
         i. **c = c² % n**
      (ii) If e_i = 1 , set c to remainder of c.m when divided by n
         i. **c = c.m %n**
   d. Store number c (encrypted form of message block M_j) in list C by appending to it

3. **C** is encrypted form of complete message M , with each element as encrypted form of corresponding message block

2) GENERATING KEY PAIR

A) Generating primenos.txt

We use the sieve method to find the list of prime numbers less than 10000 and store each of them onto a text file ‘primenos.txt’

1. Initialise list l with values True with 10001 elements except for index 0 and 1 (as checking of prime numbers only starts from 2)

2. Start from p=2 that runs while $p^2 < 10000$:

a) if l[p] = True , it confirms p as a prime:

(i) Run for loop with i from p^2 to 100001 with step value p and change all values with l[i] = False (factors of p, hence being composite)

b) p=p+1 and repeat step 2

3. Write the index values of list l having values True (the prime number positions will have True) into text file primenos.txt

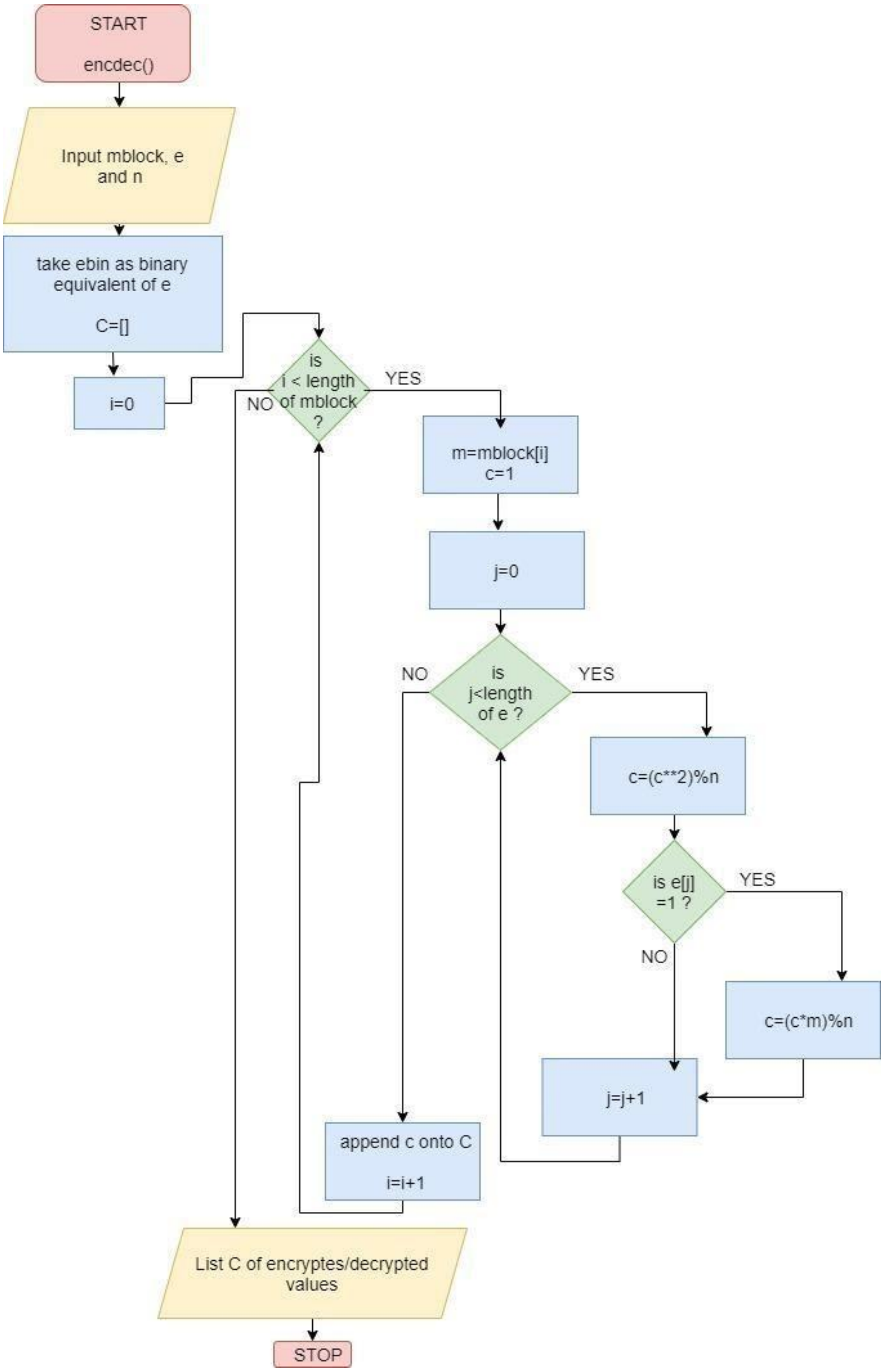
B) Generating key pair

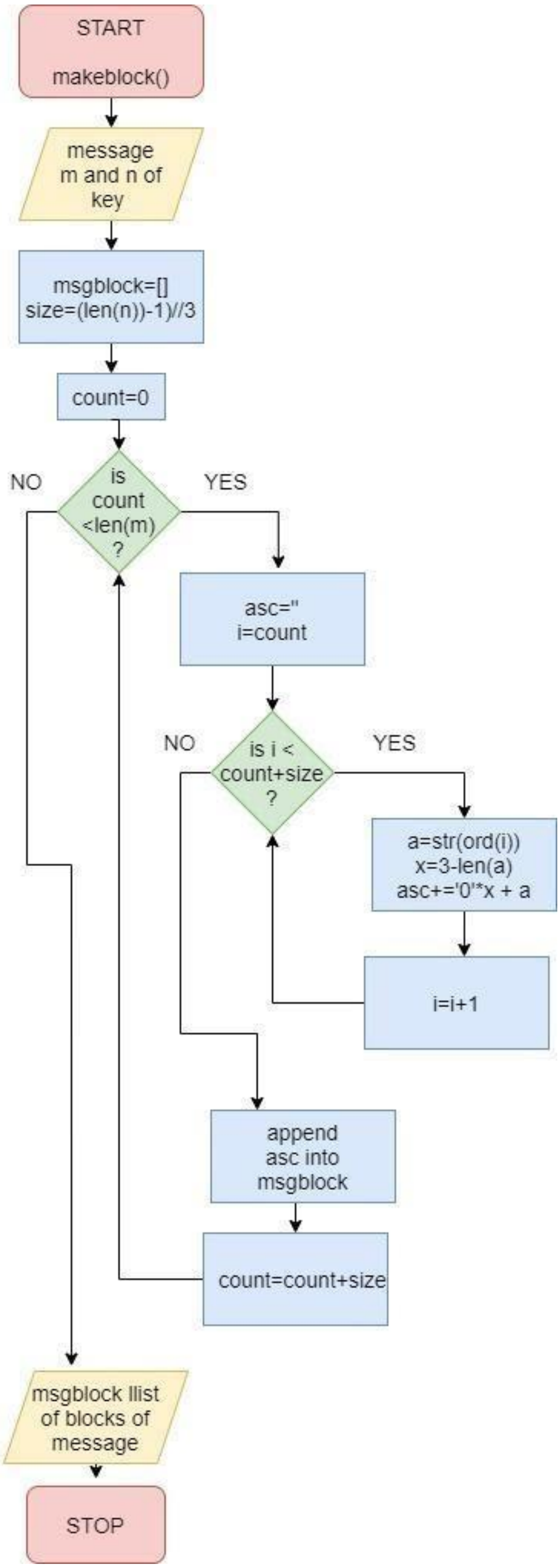
1. Choose 2 different prime numbers p and q (using primenos.txt or user input) satisfying p*q>127
2. Set n=p.q and phi=(p-1).(q-1)
3. Taking random numbers from range 3 to phi, we find number e such that it is coprime with phi

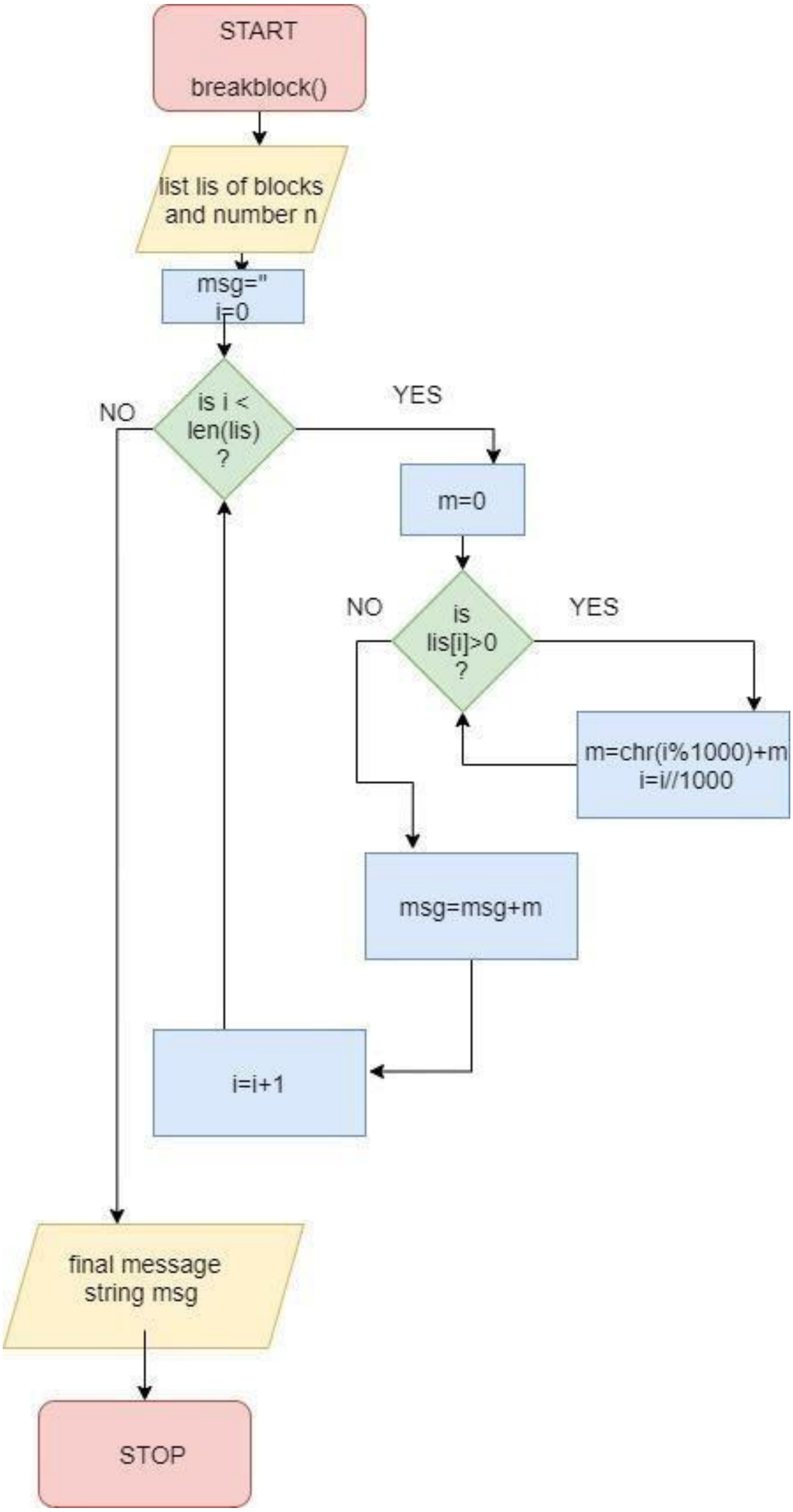
   gcd(e, phi)=1
4. Find multiplicative inverse d of e using Euclid's extended algorithm to find values x and y such that e.x + phi. y= gcd(e,phi) and set d=x
5. Our key pair (e,n) and (d,n) is formed

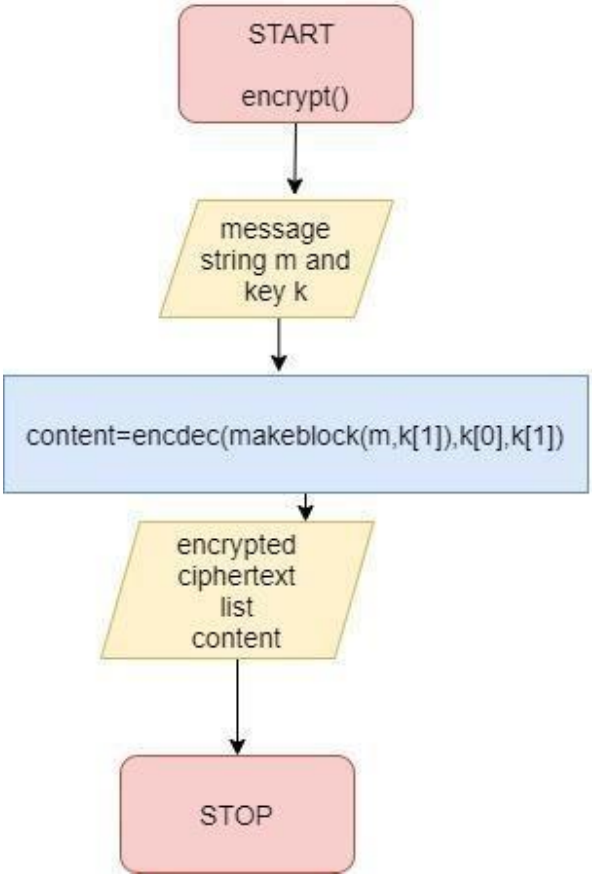
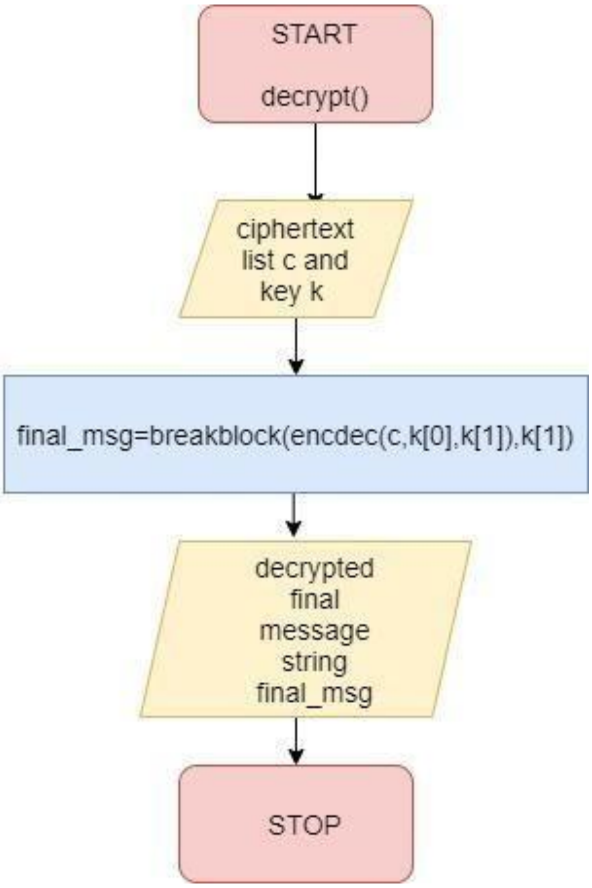
FLOWCHART

ENCRYPTION-DECRYPTION

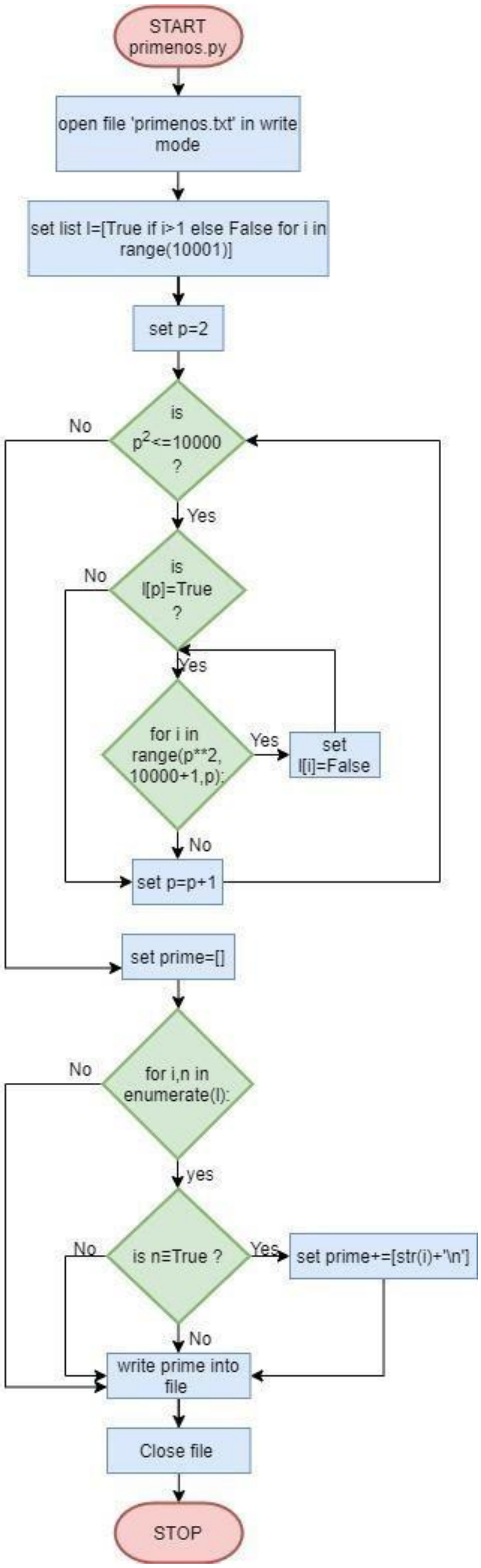



START
makeblock()
message
m and n of
key
msgblock=[]
size=(len(n))-1)//3
count=0
NO
is
count
<len(m)
?
YES
asc=''
i=count
NO
is i <
count+size
?
YES
a=str(ord(i))
x=3-len(a)
asc+='0'*x + a
i=i+1
append
asc into
msgblock
count=count+size
msgblock llist
of blocks of
message
STOP

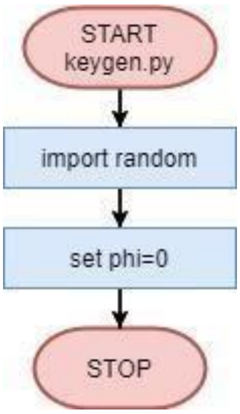

START
breakblock()
list lis of blocks and number n
msg=''
i=0
is i < len(lis) ?
YES
NO
m=0
is lis[i]>0 ?
YES
NO
m=chr(i%1000)+m
i=i//1000
msg=msg+m
i=i+1
final message string msg
STOP

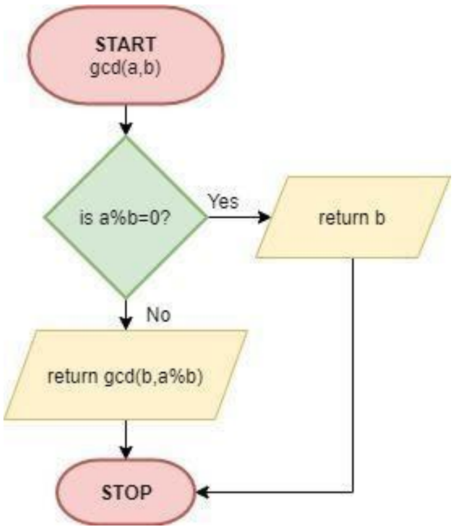

START
encrypt()
message string m and key k
content=encdec(makeblock(m,k[1]),k[0],k[1])
encrypted ciphertext list content
STOP

START
decrypt()
ciphertext list c and key k
final_msg=breakblock(encdec(c,k[0],k[1]),k[1])
decrypted final message string final_msg
STOP

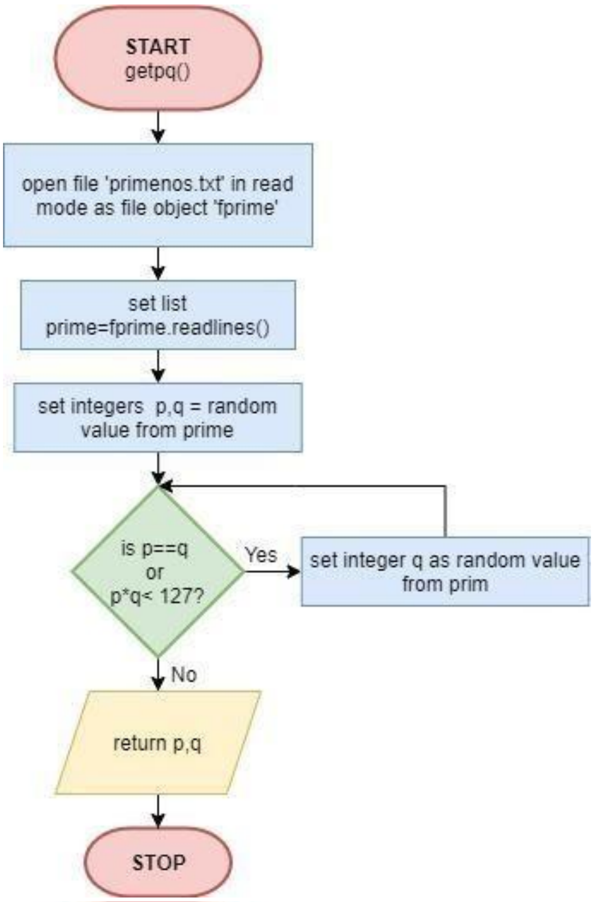
GENERATING PRIME NUMBERS

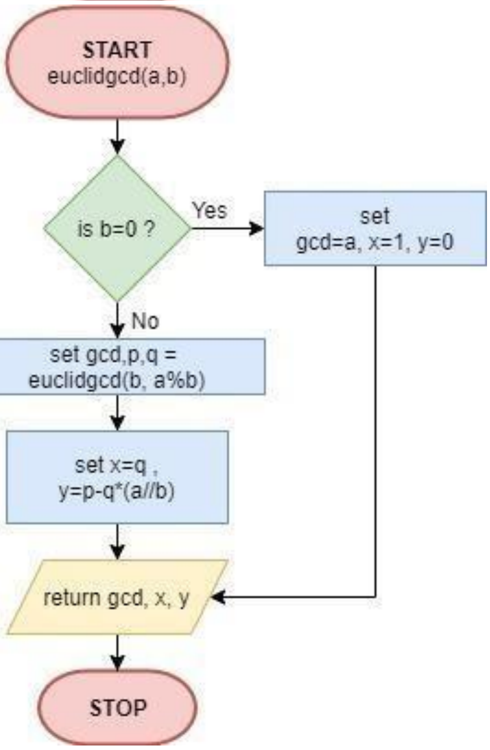


GENERATING KEY PAIR

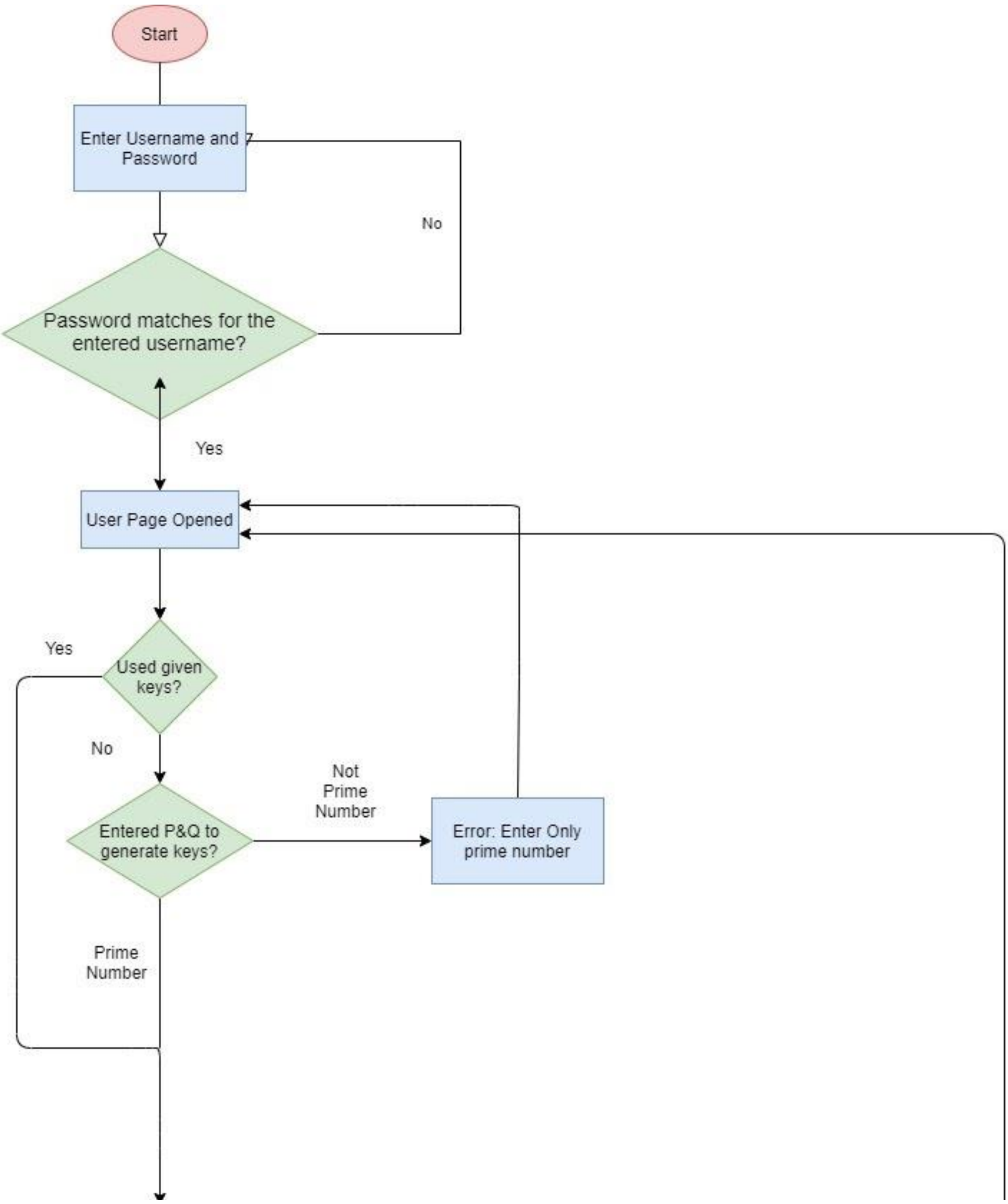


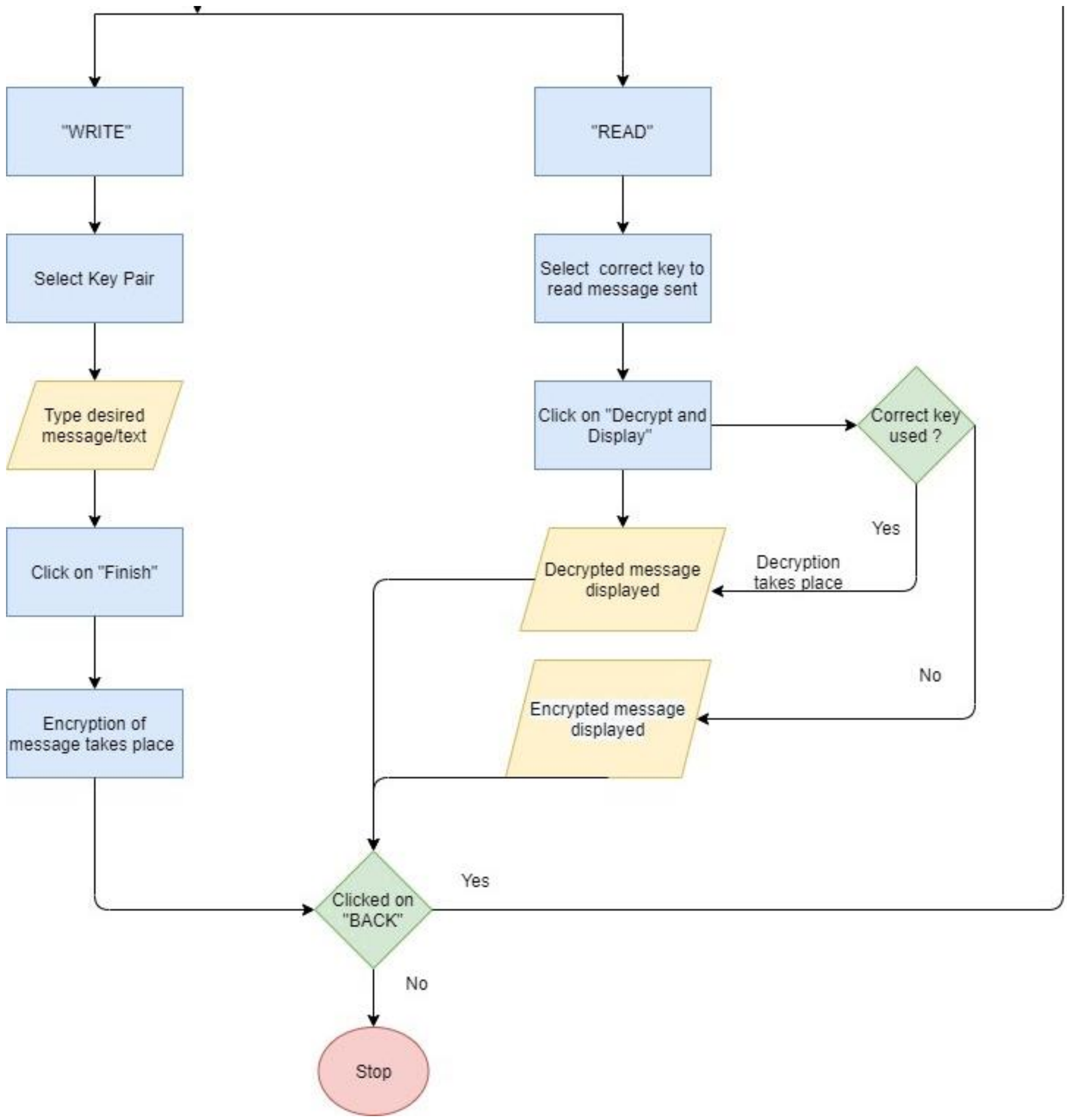



START
getpq()
open file 'primenos.txt' in read mode as file object 'fprime'
set list
prime=fprime.readlines()
set integers p,q = random value from prime
is p==q
or
p*q< 127?
Yes
set integer q as random value from prim
No
return p,q
STOP


START
euclidgcd(a,b)
is b=0 ?
Yes
set
gcd=a, x=1, y=0
No
set gcd,p,q =
euclidgcd(b, a%b)
set x=q ,
y=p-q*(a//b)
return gcd, x, y
STOP

RSA APP WORKING




"WRITE"
"READ"
Select Key Pair
Select correct key to read message sent
Type desired message/text
Click on "Decrypt and Display"
Correct key used ?
Yes
Click on "Finish"
Decrypted message displayed
Decryption takes place
No
Encryption of message takes place
Encrypted message displayed
Yes
Clicked on "BACK"
No
Stop

SOURCE CODE

ENCRYPTION-DECRYPTION

```
#encryptdecrypt.py
'''
includes
- encdec function : encrypts/decrypts the list of numbers with key given
- makeblock function : makes blocks of whole string message m
- breakblock function : reconstructs the message msg from list of blocks
- encrypt function : full encryption of message string using makeblock and encdec
                    with key k
- decrypt function : full decryption of the ciphertext back to message using encdec
                   and breakblock using key k
'''
def encdec(mblock,e,n):
    '''-inputs a list of integers as mblock
       -encrypts/decrypts each value in mblock
       -returns a list of integers called C
       corresponding to their respective
       decryptions/encryptions'''

    ebin=str(bin(e))[2:]    #to hold e in binary form
    #bin(val) converts to string of form 0bxxxxx where xxxxx represents the binary value of val
     C=[]    #to hold corresponding encrypted/decrypted values of mblock

    for m in mblock:
      m=int(m)
      c=1
      for i in ebin:
        c=(c**2)%n
        if i=='1':
            c=(c*m)%n       #evaluating c based on binary digit value of e
      C.append(c)       #final encrypted/decrypted value of element in mblock

    return C


def makeblock(m,n):

    '''breaks entire string message m into substrings of
       size based on length of n, and stores into list
       msgblock after converting to number for each
       substring using ascii number
       '''

    lenn=len(str(n))
```

```
    msgblock=[]
    size=(lenn-1)//3   #size==> determines number of characters in each block
    count=0
    while count<len(m):
        asc=''
        for i in m[count:count+size]:

            a=str(ord(i))
            x=3-len(a)         #represent of each character in a block is string of length 3
            asc+='0'*x + a         #asc==> variable holding number equivalent for each block

        msgblock.append(int(asc))       #adding the block onto final list
        count+=size
    return (msgblock)


def breakblock(lis,n):
    '''converts each element of list lis
        into the corresponding substring it represents
        using ascii value and concatenates the whole
        to reproduce the full message string msg
    '''
    msg=''   #msg==> to store final message

    for i in lis:
        m=''
        while i>0:
            m=chr(i%1000)+m         #takes every 3 digits of block which represents a character o
f message
            i=i//1000
        msg+=m             #adds the substring onto final message
     return msg

def encrypt(m,k):
    '''encrypts the message string m using the
    key k and encdec and makeblock function
    (user-defined) into ciphertext list content
    '''
    content=encdec(makeblock(m,k[1]),k[0],k[1])     #key k is tuple of form (e,n) ==> (k[0],k[1])

    return content


def decrypt(c,k):
    '''decrypts the ciphertext list c usinf the
        key k and encdec and breakblock functions
        (user-defined) into final message string
        final_msg
    '''

    final_msg=breakblock(encdec(c,k[0],k[1]),k[1])   #key k is tuple of form (e,n) ==> (k[0],k[1])
```

```
    return final_msg
```

GENERATING PRIME NUMBERS

```
'''
creates file primenos.txt and
inserts 1-4 digit prime
prime numbers are generated using sieve method
'''

file=open("primenos.txt","w") #creating file object

#create a list containing keyword False in all indexes
#except 0 and 1 where value is True
l=[True if i>1 else False for i in range(10001)]

#using sieve method to generate prime numbers
p =2
while p**2 <=10000:
    if l[p]==True:
     for i in range(p**2,10000+1,p): #iterating from p^2 to 10,000
        l[i]=False #positions of composite numbers assigned value False
    p+=1

prime=[] #list containg primes
for i,n in enumerate(l):
    if n:
        prime+=[str(i)+'\n'] #appending list with prime numbers

file.writelines(prime) #writing prime numbers into file
file.close()
```

GENERATING KEY PAIR

```
#keygen.py

'''
Generate RSA Key pair
includes
- getpq function : choosing random primes p,q
- gcd function : returns greatest common divisor of integers a,b
- euclidgcd function : extended Euclidean algorithm to calculate d
- main function : to generate key pair using above functions
'''
```

```
import random

def gcd(a,b):
    '''
    Performs long division method to find gcd of integers a,b
    Returns gcd
    '''
    if a%b==0:
        return b
    else:
        return gcd(b,a%b)

def euclidgcd(a,b):
    '''
    Performs the extended Euclidean algorithm
    Returns the gcd, coefficient of a, and coefficient of b
    in eqn gcd(a,b) = ax + by
    '''
    if b==0:
        gcd,x,y=a,1,0
    else:
        gcd,p,q=euclidgcd(b,a%b)
        x=q
        y=p-q*(a//b)
    return gcd,x,y

def getpq():
    '''
    opens file primenos.txt and
    choose two distinct numbers p,q from it and
    returns them
    '''
    fprime=open('primenos.txt','r') #creating file object
    prime=fprime.readlines() #reading file

    p,q=int(random.choice(prime)),int(random.choice(prime)) #choosing random primes p,q
    while p==q or p*q<127: #ensure primes are distinct and their product p*q>=127
        q=random.choice(prime)

    return p,q

def main(p=0,q=0):
    '''
    generates RSA key pairs
    may take input for primes p,q or chooses random primes
    computes RSA key pair using these primes
    returns RSA key pairs (e,n) and (d,n) as values of e,d,n
    '''
    global phi
    if p+q==0: #check if user input is given
```

```
        p,q=getpq() #else chooses random primes p,q
    n=p*q #computen
    phi=(p-1)*(q-1) #compute phi(n)
    e=3 #set default value of e
    while gcd(e,phi)!=1: #ensure that gcd(e,phi) is 1
        e=random.randrange(3,phi) #else choose random value of e such that 3<e<phi

    a,d,b=euclidgcd(e,phi) #getting value of d = e^(-1)
    d=d%phi #make d part of primary ring of integers (mod phi)
    return e,d,n #return key pair values

phi=0 #to store phi(n)
```

USER INTERFACE

```
#importing modules/packages
import tkinter as tk
import keygen as kg
import encryptdecrypt as ed

from tkinter import tix

class user:
    def __init__(self,name,colour,pd):
        self.name=name
        self.colour=colour
        self.pd=pd
        self.kpu=None
        self.kpr=None
        self.newkeypair()


    def userhomepg(self):

        def checkpq():
            '''check validity of p and q for key generation and
                and perform operations accordingly'''

            pu,qu=e1.get(),e2.get() #get values from entry box
            f=open('primenos.txt') #get prime numbers
            content=f.read().split() #list containing primes

            if pu==qu=='use random':
                self.newkeypair()
                l4['text']=str(self.kpu)
                l5['text']=str(self.kpr)
            elif pu in content and qu in content and int(pu)*int(qu)>127 and pu!=qu:
                self.newkeypair(pu,qu)
                l4['text']=str(self.kpu)
```

```
            l5['text']=str(self.kpr)

        else:
            #creating an error window
            err=tk.Toplevel(nw)
            err.title('ERROR')
            err.geometry('250x70')
            err_label=tk.Label(err,text='INVALID INPUT\np and q must be 2 primes\np*q>127\
np cannot be equal to q')
            err_label.pack()
            #tk.Message(nw,text='Invalid p and q values. Try again.')

    #creating window definitions
    nw = tix.Toplevel(root)
    nw.state('zoomed')
    nw.title(self.name)
    nw.configure(bg=self.colour)
    nw.geometry('500x400')

    def createbal(widget,msg):
        b=tix.Balloon(nw)
        b.bind_widget(widget,balloonmsg=msg)

    def helpbtn():
        hel=tk.Toplevel(nw)
        hel.title('HELP')
        hel_label=tk.Label(hel,text='''Welcome to your homepage !
Your default key pair has been generated and displayed

Here's what you can do :

'Keys Available' - Displays the keys available to you

'ENTER P AND Q' - To create new key pair
Enter unique prime numbers p and q into entry box
or use the randomly generated primes simply by
clicking the 'CREATE NEW KEY' button

'Write' - To write text message and encrypt using selected key

'Read' - To decrypt using selected key and read text message''')
        hel_label.pack()

    #on clicking help window will open which gives instructions on usage
    help1=tk.Button(nw, text = 'HELP BUTTON\n    (?)',bg='orange',command=helpbtn)
    help1.place(x=900,y=50)

    #labels to display keypairs of self
    l1=tk.Label(nw,bg='red',text='YOUR KEY PAIR',font='20')

    l2=tk.Label(nw,bg='green',fg='white',text='Public Key')
```

```
        createbal(l2,'Key available to all users')

        l3=tk.Label(nw,bg='green',fg='white',text='Private Key')
        createbal(l3,'Key available to only you')

        l4=tk.Label(nw,text=str(self.kpu))#user's public key
        l5=tk.Label(nw,text=str(self.kpr))#user's private key

        l1.pack(padx=10)
        l2.place(x=500, y=50)
        l3.place(x=710,y=50)
        l4.place(x=500,y=80)
        l5.place(x=710,y=80)

        #for keys available to user
        keys=['Pr '+self.name+str(self.kpr),'Pu '+userA.name+str(userA.kpu),'Pu '+userB.name+
str(    userB.kpu),'Pu '+userC.name+str(userC.kpu)]
        variable = tk.StringVar(nw)
        variable.set('Keys Available') # default value
        dropd=tk.OptionMenu(nw,variable,*keys)
        createbal(dropd,'See the keys available to you')
        dropd.pack(pady=10)

        #user enters key
        la=tk.Label(nw,bg='blue',fg='white',text='ENTER P AND Q')
        e1=tk.Entry(nw)
        e1.insert(tk.END,'use random') #default val
        createbal(e1,'Or enter your own prime numbers')
        e2=tk.Entry(nw)
        e2.insert(tk.END,'use random') #default val
        createbal(e2,'Or enter your own prime numbers')
        extralab=tk.Label(nw,bg=self.colour)
        extralab.pack(pady=25)
        la.pack(pady=10)
        e1.pack(pady=5)
        e2.pack(pady=5)

        create=tk.Button(nw, text = 'Create new key',bg='pink',command=checkpq)#on clicking
butt    on new key gets generated with either random values or user defined values
        createbal(create,'click to create')
        create.pack(pady=10)

        #to write message to someone
        button4 = tk.Button(nw, text = 'WRITE', width = 25,command = self.write)#opens new p
age where user can write a message
        button4.place(x=547,y=350)
        createbal(button4,'write a message')

        #to read message from someone
        button5 = tk.Button(nw, text = 'READ', width = 25,command = self.read)#opens a page
where user can read a message
```

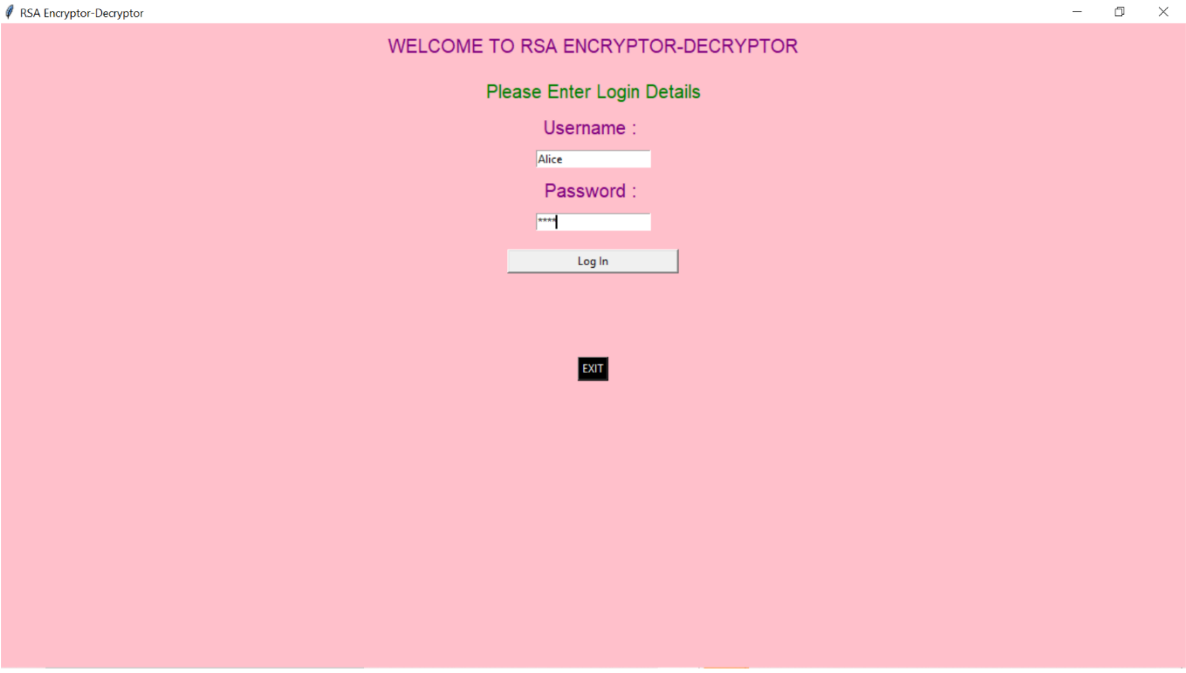
```
        button5.place(x=547, y=400)
        createbal(button5,'read a message ')

        #go to previous window
        bexit=tk.Button(nw,text='HOME',fg='white',bg='black',command=nw.destroy)#redirects t
o login page
        bexit.pack(padx=100,pady=150)

    def newkeypair(self,p=0,q=0):
        ''' creates new key pair and assigns to self'''
        e,d,n=kg.main(int(p),int(q))
        self.kpu,self.kpr = (e,n),(d,n)#key pairs for user

    def write(self):
        #opening file object
        nw = tk.Toplevel(root)
        nw.state('zoomed')
        nw.configure(bg=self.colour)

        #dropdown
        l=[self.kpr,userA.kpu,userB.kpu,userC.kpu]
        keys=['Pr '+self.name+str(l[0]),'Pu '+userA.name+str(l[1]),'Pu '+userB.name+str(l[2]),'Pu
 '+userC.name+str(l[3])]
        variable = tk.StringVar(nw)
        variable.set('Send message using') # default value
        dropd=tk.OptionMenu(nw,variable,*keys)#opens a drop down box showing available ke
ys to the user
        dropd.pack(pady=10)

        #text
        text=tk.Text(nw,height=20,width=50)
        text.pack()

        def getinfo():
            f=open('textmessage.txt','w+')
            msg=text.get('1.0','end')
            k=l[keys.index(variable.get())]  #k==> key
            file_con=ed.encrypt(msg,k)#message gets encrypted
            f.write(str(file_con))#encrypted message gets stored in a file
            f.close()

        #close file
        close_button=tk.Button(nw,text='finish',bg='pink',command=getinfo)
        close_button.pack(pady=20)

        #go to previous window
        bexit=tk.Button(nw,text='USER PAGE', fg='white', bg='dark green',command=nw.destro
y)
        bexit.pack(padx=100,pady=100)

    def read(self):
```

```
        #window def
        nw = tk.Toplevel(root)
        nw.state('zoomed')
        nw.configure(bg=self.colour)
        button1 = tk.Button(nw, text = 'Public key for '+self.name, width = 25, bg='orange')
        button1.pack(pady=10)

        #opening file object
        la=tk.Label(nw)

        #dropdown
        l=[self.kpr,userA.kpu,userB.kpu,userC.kpu]
        keys=['Pr '+self.name+str(l[0]),'Pu '+userA.name+str(l[1]),'Pu '+userB.name+str(l[2]),'Pu
 '+userC.name+str(l[3])]
        variable = tk.StringVar(nw)
        variable.set(keys[0]) # default value
        dropd=tk.OptionMenu(nw,variable,*keys)
        dropd.pack(pady=10)

        def decndis():
            f=open('textmessage.txt','r')
            k=l[keys.index(variable.get())]  #k==> key
            file_con=f.read()
            msg=ed.decrypt(file_con[1:-1].split(','),k)
            la['text']=msg#the decrypted message is displayed
            la.place(x=600,y=150)
            f.close()

        #decrypt button
        close_button=tk.Button(nw,text='Decrypt and Display',bg='pink',command=decndis)
        close_button.pack(pady=10)

        #go to previous window
        bexit=tk.Button(nw,text='USER PAGE',fg='white',bg='dark green',command=nw.destroy
)
        bexit.pack(padx=100,pady=100)

def open_txt():
    #pass
    text_file = open('Test.txt','r')
    content=text_file.read()
    text.insert(END,content)
    text_file.close()

#user object definitions
userA=user('Alice','light green','a123')
userB=user('Bob','light blue','b123')
userC=user('Oscar','yellow','o123')
user_pd={'Alice':userA,'Bob':userB,'Oscar':userC}
```

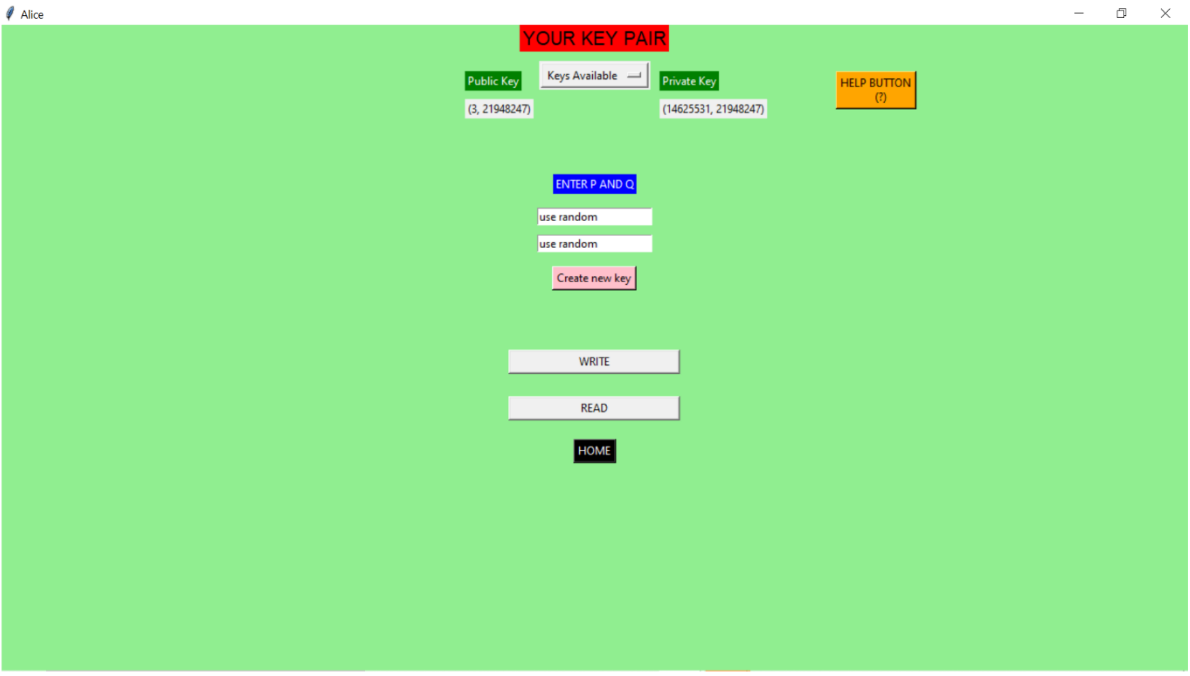
```
def checklog():
    try:
        if user_pd[e1.get()].pd==e2.get():
            l5.pack_forget() # <=====
            user_pd[e1.get()].userhomepg()#directs to user's page
    except KeyError:
        l5['text']='Invalid username/password'
        l5.pack()


#ROOT WINDOW

root=tix.Tk()
root.title('RSA Encryptor-Decryptor')
root.state('zoomed')
root.configure(bg='pink')
root.geometry('500x400')
l1=tk.Label(text='WELCOME TO RSA ENCRYPTOR-
DECRYPTOR',fg='purple',font='20',bg='pink')
l1.pack(pady=10)
l2=tk.Label(text='Please Enter Login Details',fg='green',font='20',bg='pink')
l2.pack(pady=10)
l3=tk.Label(text='Username : ',fg='purple',font='20',bg='pink')
l3.pack()
e1=tk.Entry()
e1.pack(pady=10)
l4=tk.Label(text='Password : ',fg='purple',font='20',bg='pink')
l4.pack()
e2=tk.Entry(show='*')
e2.pack(pady=10)
l5=tk.Label(fg='purple',font='20',bg='pink')

b1=tk.Button(root,text='Log In', width=25,command = checklog)#checks the user and passw
ord are matching or not
b1.pack(pady=10)

#exit button
bexit=tk.Button(root,text='EXIT',fg='white',bg='black',command=root.destroy)#closes the pro
gram
bexit.pack(padx=80,pady=80)
```

OUTPUT

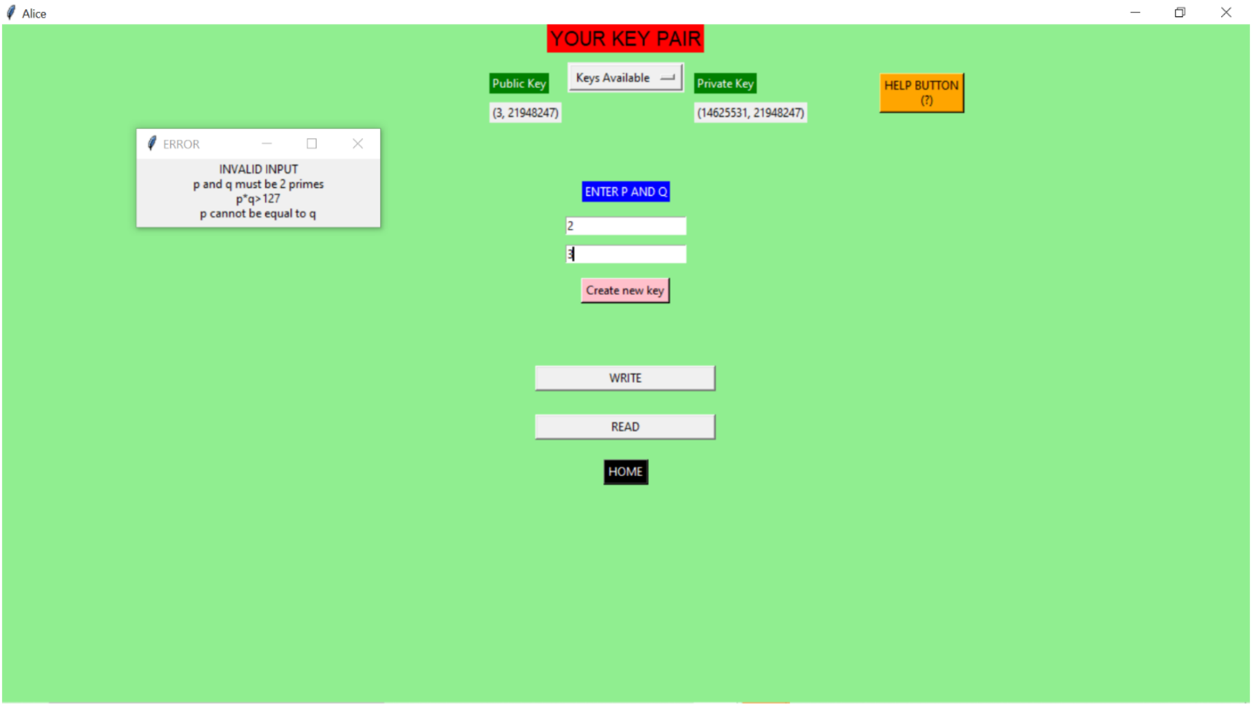
1. User Login Page



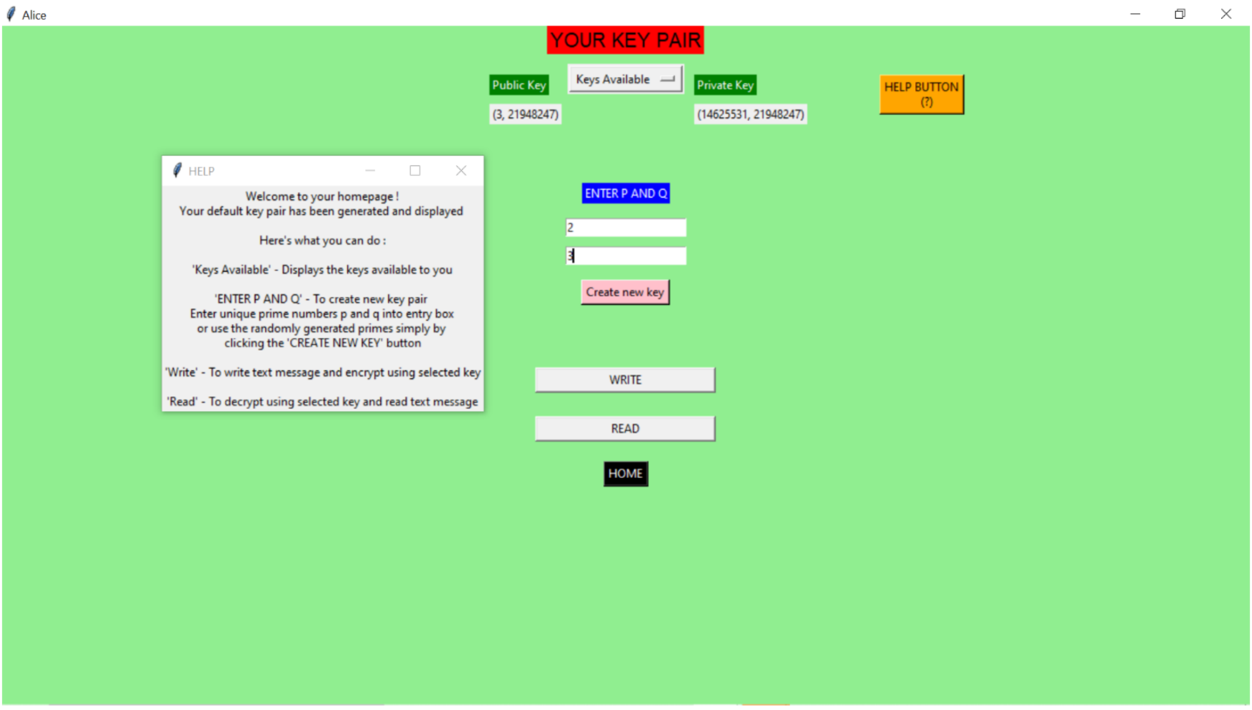
2. User Page (Alice)



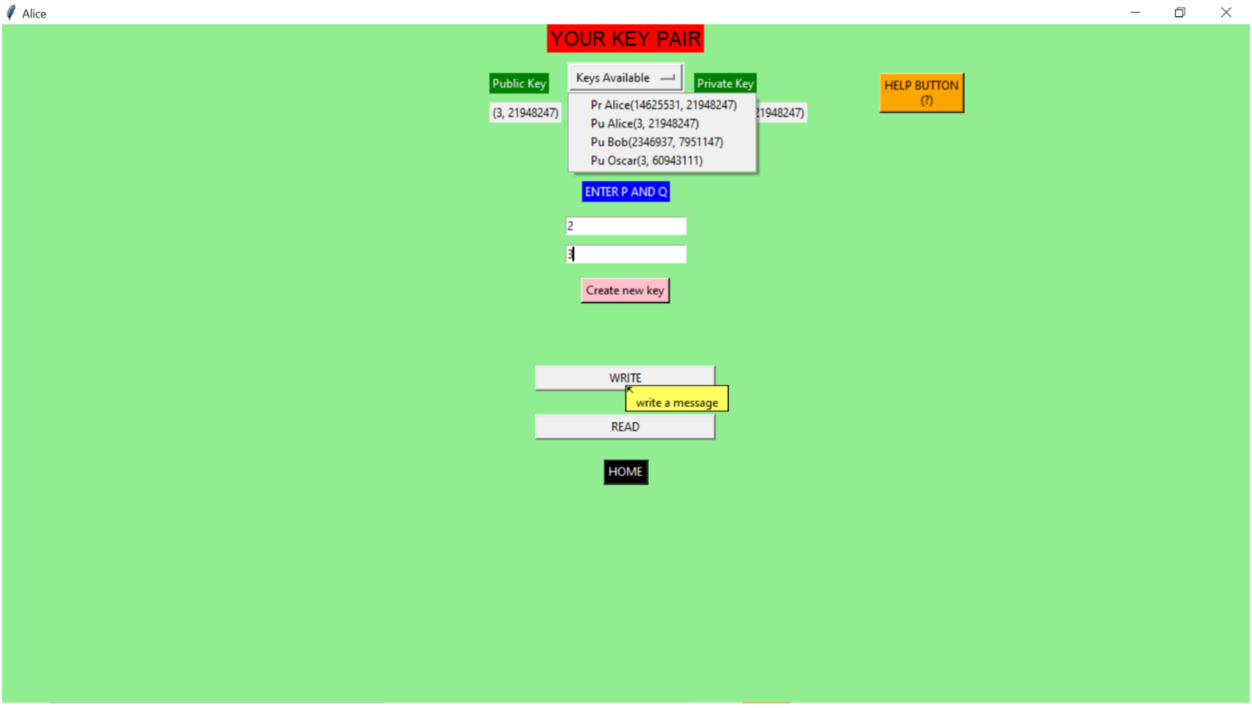
3. Error Box opened if user defined values of P and Q are wrong



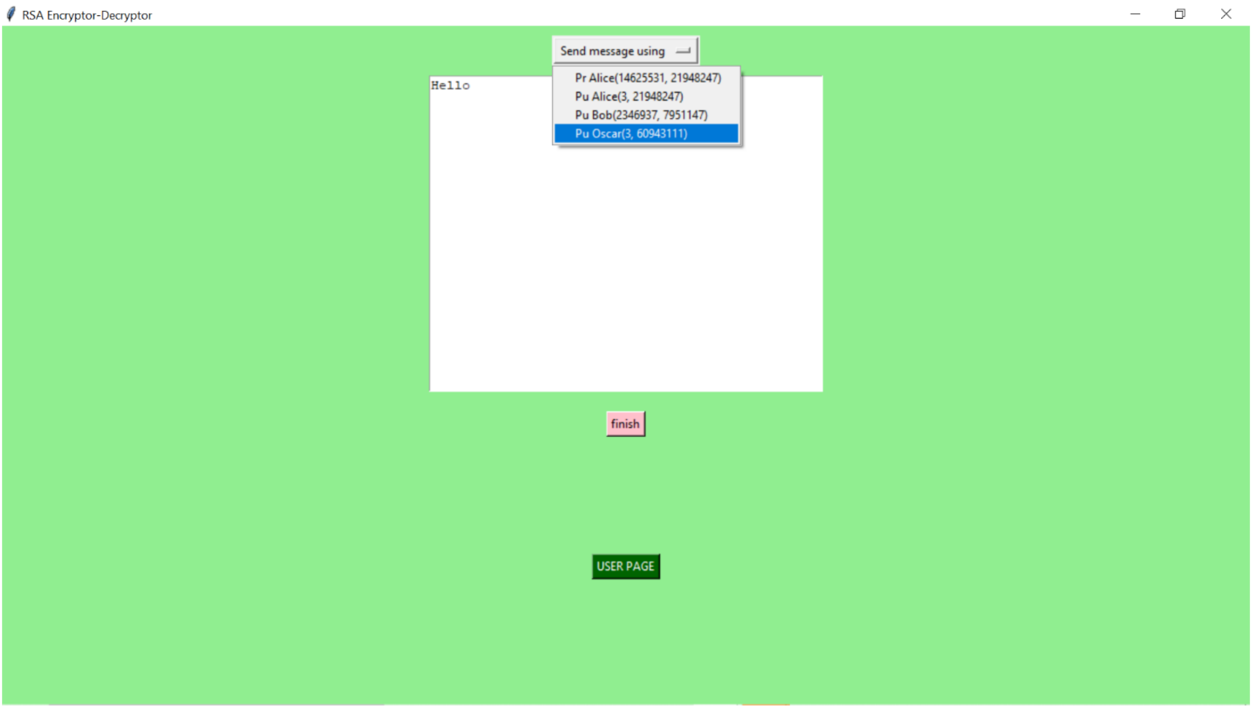
4. Help box opened



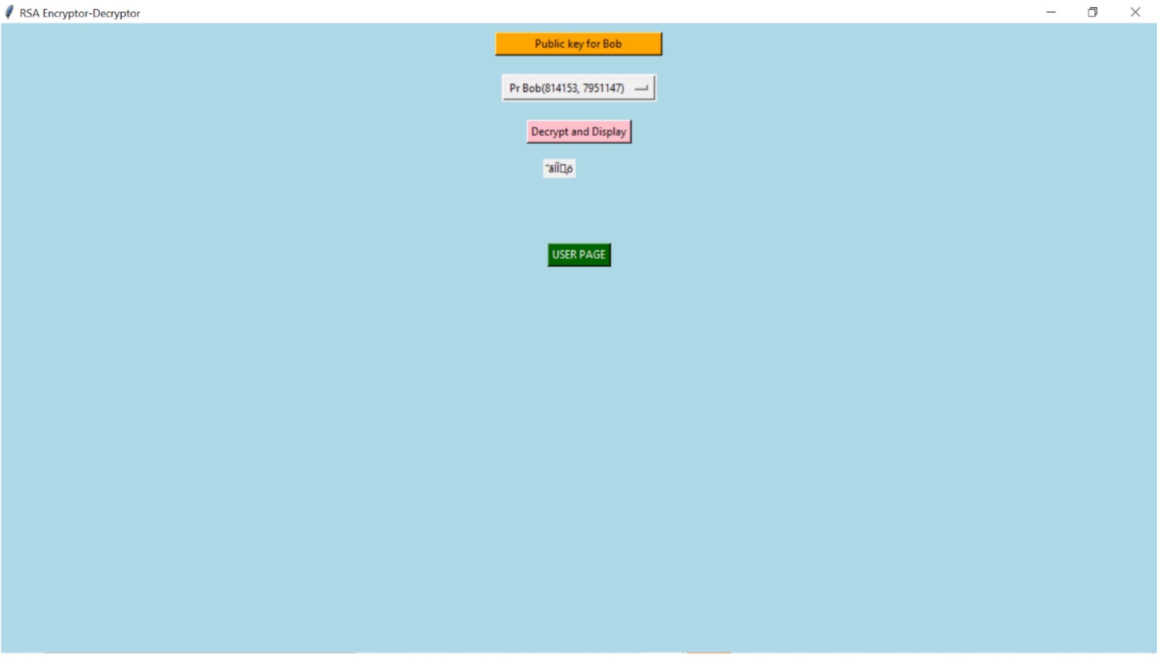
5. Drop down box showing key pairs



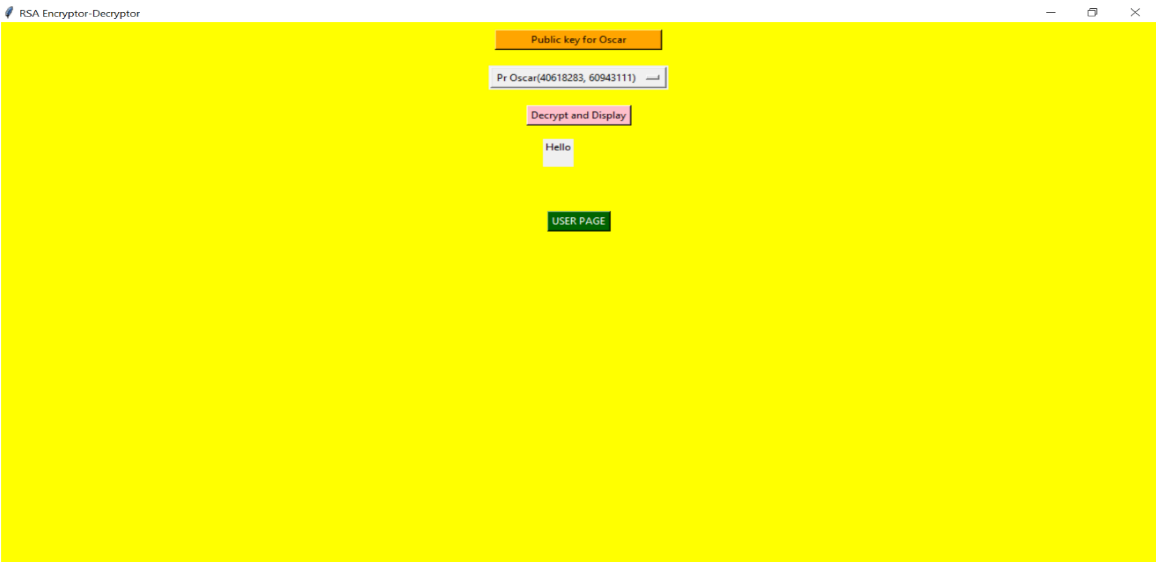
6. Write window (Sending message to Oscar)



7. Read window of another user (Bob). You can see that once he tries to access the encrypted message he cannot do so as the message was meant to be sent for Oscar.



8. Read window (Oscar). We see that on clicking the required key pair he was able to decrypt and see Alice's message



CONSTRAINTS AND SCOPE

Here, we shall discuss the limitations and scope of our project in the future

- Use in digital signatures – RSA cryptology is generally used in creation of digital signatures (digital signatures are mathematical schemes used for verifying the authenticity of a digital message). Our program is a very simplified version made for showing the use of RSA in encryption-decryption of simple text messages.

- The key size we are using in our program is very small compared to the actual key size used in proper RSA encryption. Because it is so large it would be stored in a file instead of storing it in memory like our program.

- Our program only caters to a limited number of pre-saved users (i.e. 3). So, we can try adding a feature where a greater number of users can create and use their own id which will be saved in the system

Our program uses one system for all users and the messages are usually sent and stored in a text file. In future we would like to move this platform to the internet where a large number of users can communicate safely all over the globe.

BIBLIOGRAPHY

Cryptography and Network Security by William Stallings

https://people.csail.mit.edu/rivest/Rsapaper.pdf

https://www.tutorialspoint.com/cryptography/public_key_encryption.htm

https://www.cloudflare.com/learning/ssl/how-does-public-key-encryption-work/

https://www.ssh.com/ssh/