SECURE CHAT APPLICATION USING RSA ALGORITHM

CRYPTOGRAPHY PROJECT

CSE1007 FALL SEMESTER-2020-21 E+TE SLOT

Submitted by

Lakky Setty Koushik 19BCE7279

Namala Sahithi 19BCN7042

Guided By

Prof. GARIMA SINGH

ABSTRACT

The RSA is the most widely deployed public-key cryptosystem and is used for both encryption and digital signature. It is commonly used in securing ecommerce and e-mail, implementing virtual private networks and providing authenticity of electronic documents. It is implemented in most Web servers and browsers, and present in most commercially available security products. In fact, the ubiquity of RSA has placed it at the heart of modern information security. It would not be an overstatement to say that Internet security relies heavily on the security properties of the RSA cryptosystem. Since its invention in 1977, the RSA cryptosystem has been extensively analyzed for vulnerabilities. While no devastating attack has ever been found, years of cryptanalysis of RSA have given us a broad insight into its properties and provided us with valuable guidelines for proper use and implementation. In this project, we are applying RSA Algorithm in udp chat application using socket programming for secure transmission of data. Large Prime numbers are generated to have a secured cipher text.

INTRODUCTION CRYPTOGRAPHY:

Cryptography from Greek , "hidden, secret" respectively is the practice and study of techniques for secure communication in the presence of third parties (called adversaries).More generally, it is about constructing and analyzing protocols that overcome the influence of adversaries and which are related to various aspects in information security such as data confidentiality, data integrity, authentication, and non-repudiation. Modern cryptography intersects the disciplines of mathematics, computer science, and electrical engineering. Applications of cryptography include ATM cards, computer passwords, and electronic commerce. Cryptography prior to the modern age was effectively synonymous with encryption, the conversion of information from a readable state to apparent nonsense. The originator of an encrypted message shared the decoding technique needed to recover the original information only with intended recipients, thereby precluding unwanted persons to do the same. Since World War I and the advent of the computer, the methods used to carry out cryptology have become increasingly complex and its application more widespread. Modern cryptography is heavily based on mathematical theory and computer science practice. Cryptographic algorithms are designed around computational hardness assumptions, making such algorithms hard to break in practice by any adversary. It is theoretically possible to break such a system but it is infeasible to do so by any known practical means. These schemes are therefore termed computationally secure; theoretical advances and faster computing technology require these solutions to be continually adapted. There exist information theoretically secure schemes that provably cannot be broken even with unlimited computing power—an example is the one-time pad—but these schemes are more difficult to implement than the best theoretically breakable but computationally secure mechanisms

KEY BASICS: The two components required to encrypt data are an algorithm and a key. The algorithm generally known and the key are kept secret. The key is a very large number that should be impossible to guess, and of a size that makes exhaustive search impractical. In a symmetric cryptosystem, the same key is used for encryption and decryption. In an asymmetric cryptosystem, the key used for decryption is different from the key used for encryption.

KEY PAIR: In an asymmetric system the encryption and decryption keys are different but related. The encryption key is known as the public key and the decryption key is known as the private key. The public and private keys are known as a key pair. Where a certification authority is used, remember that it is the public key that is certified and not the private key. This may seem obvious, but it is not unknown for a user to insist on having his private key certified!

RSA Algorithm for Encryption Decryption:

RSA is the most widely used and tested public-key cryptosystem. It stands for Rivest, Shamir, and Adleman. RSA was invented in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman. It is based on a very simple numbertheoretical idea, and yet it has been able to resist all cryptanalytic attacks. The idea is a clever use of the fact that, while it is easy to multiply two large primes, it is extremely difficult to factorize their product.Thus, the product can be publicized and used as the encryption key. The primes themselves cannot be recovered from the product and are used for decryption. Two points need to be borne in mind however, while dealing with the RSA system there is no formal proof whatsoever that factorization is intractable or is intractable in the special case needed for RSA, and that factorization is needed for the cryptanalysis of the RSA

LITERATURE SURVEY

Chaotic synchronization cryptosystems combined with RSA encryption algorithm.

Chaotic synchronization has suffered from demodulating information by hackers via the public channel. Different methods have been developed to hide the contents of a message using chaotic signals. Nevertheless, many studies have shown that most of the existing methods are unreliable in the aspect of security. According to the above, in order to enhance the strength of the cryptosystem and provide greater security, we carry out double encryption which combines chaotic synchronization with RSA algorithm. In this study, a design methodology for neural-network (NN)-based secure communications in multiple time-delay chaotic (MTDC) systems is proposed to obtain double encryption via RSA algorithm and chaotic synchronization. RSA algorithm is an asymmetric encryption and its strength is based on the tremendous difficulty of factorizing two large prime numbers. Accordingly, it takes much more time to process RSA algorithm if the length of the key is much longer. A shorter key is used to decrease the processing time of RSA algorithm in this work. Nevertheless, it will result in a lower security of the cryptosystem. In order to enhance the strength of the Cryptosystem, we carry out double encryption which combines chaotic synchronization with RSA Algorithm. Moreover, an IGA is proposed in this study to effectively overcome the weakness of the traditional GA. On the basis of the IGA, a fuzzy controller is synthesized to not only realize the exponential Synchronization, but also achieve optimal High performance by minimizing the disturbances attenuation level.With this we can confirm that the IGA is better than traditional GAs in terms of both globalization and convergence rates.

Secure and robust digital image watermarking scheme using logistic and RSA encryption

In the era of big data and networking, it is necessary to develop a secure and robust digital watermarking scheme with high computational efficiency to protect copyrights of digital works. However, most of the existing methods focus on robustness and embedding capacity, losing sight of security or requiring significant computational resources in the encryption process. This paper proposed a new digital image watermarking model based on scrambling algorithm Logistic and RSA asymmetric encryption algorithm to guarantee the security of the hidden data at the foundation of large embedding capacity, good robustness and high computational efficiency. The experiments involved applying the encryption algorithms of Logistic and RSA to the watermark image and performing the hybrid decomposition of Discrete wavelet transform (DWT) and Singular Value Decomposition (SVD) on the host image, and the watermark was embedded into the low-frequency sub-band of the host. The values of PSNR(Peak Signal to Noise Ratio) and NCC (National Coordinating Center) were measured to estimate the imperceptibility and robustness of the proposed watermarking scheme, and the CPU running time was recorded to measure the complexity of the proposed main algorithm in execution time. Experimental results showed the superiority of the proposed watermarking scheme.

Improvement of trace-driven I-Cache timing attack on the RSA algorithm

The previous I-Cache timing attacks on the RSA algorithm which exploit the instruction path of a cipher are mostly proof-of-concept, and it is harder to put them into practice than D-Cache timing attacks. We propose a trace-driven timing attack model on the RSA algorithm via spying on the whole I-Cache, instead of the partial instruction cache to which the multiplication function mapped, by analyzing the complications in the previous I-Cache timing attack on the RSA algorithm. Then, an improved analysis algorithm of the exponent using the characteristic of the window size in SWE algorithm is provided, which could further reduce the search space of the key bits than the former. We further demonstrate how to recover the private key d from the scattered known bits of dp and dq, through demonstrating some conclusions and validating it by experimentation. In addition, an error detection mechanism to detect some erroneous decisions of the operation sequences is provided to reduce the number of the erroneous recovered bits, and improve the precision of decision. We implement an I-Cache timing attack on RSA of OpenSSL in a practical environment, the experimental results show that the feasibility and effectiveness of I-Cache timing attack can be improved

Design of RSA processor for concurrent cryptographic transformations

The performance of RSA depends strongly on the competent implementation of modular multiplication and modular exponentiation. Performance can be improved in three ways: (i) by reducing the frequency of modular multiplications; (ii) by reducing the time required to evaluate modular multiplication; (iii) by increasing the RSA cores. This work proposes enhancements to the Montgomery Multiplication and also to Square & Multiply algorithm. Bit Forwarding 1-bit (BFW1) algorithm has been implemented to evaluate modular exponentiation that resulted in 11.11% improvement in throughput, and 1.90% reduction in power consumption. A Dual-core RSA processor with a hardware scheduler has been designed for performing concurrent cryptographic transformations to attain better throughput without increasing the frequency.

The proposed hardware scheduler is able to increase the throughput of 95.85% for MSM algorithm and 117.61% for BFW1 for 1024-bit key with reference to the MME42_C2 algorithm. The results have been verified for a key of length 1024-bits, up to 32-cores. This is scalable, by proportionally increasing BRAM and priority queue size.

Fast Cloud-RSA Scheme for Promoting Data Confidentiality in the Cloud Computing

Confidentiality is among the most important obstacles for widespread adoption of cloud services. In fact, researchers have invented a new promising form of encryption, homomorphic encryption, that can be considered as an effective way to overcome these obstacles. Since cloud computing environment is more threatened by attacks and since cloud consumers often use resource-constrained devices to access to cloud computing services, the homomorphic encryption schemes must be promoted, in terms of security level and running time, to work efficiently. At EMENA-TSSL'16, we boosted the standard RSA encryption scheme at security level, Cloud-RSA. In this paper, we propose a new fast variant of the Cloud-RSA scheme to speed up its algorithms. The proposed variant uses a modulus of the form $N = prqs$ for $r, s \geq 2$ and employs Hensel lifting and Chinese remaindering to decrypt. Simulation results show that the proposed variant gives a large speed up over the Cloud-RSA scheme while preserving a prescribed security level

An Enhanced and Secured RSA Key Generation Scheme (ESRKGS)

In this article, we do the cryptanalysis of the Enhanced and Secured RSA Key Generation Scheme (ESRKGS) given in Thangavel et al. (2015). The authors claimed that ESRKGS is a highly secure and not easily breakable scheme compared to the traditional RSA scheme. In their scheme, the public key n is a product of two large prime numbers, but the values of Encryption E and Decryption D keys are based on the product of four large prime numbers, N, which is a multiple of n. They claimed that even if someone factorizes n, in order to obtain the private key D, brute force must be used to obtain the other two primes.

We prove that we can use an alternative private key D1 to break the system. Someone does not need to employ brute force to break the system when n is factored. Hence, ESRKGS has the same security level as the traditional RSA.

PURPOSE

Its purpose is to have a secured transmission of information from one individual to another securely using a secured method. The programs developed for this project use sockets for the creation of a connection between the receiver and sender. To send the data, special protocols like TCP and UDP are used. For safe transmission of data, the message is changed into cipher text with the help of RSA Algorithm.

SCOPE AND OBJECTIVE

Our work in this project is focused primarily on the implementation of RSA along with socket programming. For efficient implementation we have used various inbuilt functions and developed new functions for generation of large prime numbers for generation of keys which will lead to more secure data encryption.

TOOLS AND ENVIRONMENT USED

Software Requirement

1) Language : Python

2) Platform : Windows

3) Client : Client designed using Python

4) Server : Server designed using Python

PROPOSED WORK

Working of RSA Algorithm:

The working of the RSA can be explained in 3 stages:

1. Public and Private Keys: Take two large prime numbers p and q (of the order of a few hundred bits).

Compute their product n. Also compute the Euler function - Φ (n) = (p-1) (q-1)

2. Choose a large random number d (d > 1) such that (d; Φ (n)) = 1 (i.e., d and Φ (n) are relatively prime).

3. Compute the number e, 1 < e < Φ (n) such that ed ≡ 1(mod Φ (n)) (i.e., ed -1 is divisible by (p - 1) (q - 1)). An introduction to some terminology will be appropriate at this juncture.

4. Encrypt the message using the public key. 5. Decrypt the message using private key. Terminology: n: Modulus or Key d: Private or Decryption Exponent e: Public or Encryption Exponent (n; e): Public Key (n; d): Private Key p, q, Φ (n),d : form the Secret Trapdoor (p and q may be kept with the private key or destroyed).

Terminology:

n: Modulus or Key

d: Private or Decryption Exponent

e: Public or Encryption Exponent (n; e): Public Key

(n; d): Private Key

p, q, Φ (n),d : form the Secret Trapdoor (p and q may be kept with the private key or destroyed).

Important Features of RSA:

While studying the working of the RSA system, we need to note the following:

Encryption and authentication takes place without sharing of private keys: each person uses only other people's public keys and his/her own private key.

Anyone can send an encrypted message or verify a signed message, using only public keys, but only someone in possession of correct private keys can decrypt or sign a message.

Modular Exponentiation: The computation of (ar; mod n) is done using a method that is faster than repeatedly multiplying a by itself. We use squaring. After each squaring, reduction modulo n is done. So we never encounter numbers greater than n2. Thus (ar; mod n) can be computed in O(log r) time.

Choice of Primes:

The primes p and q need to be random primes and not some primes contained in some table of primes (to factorize, one can always check through the table). p and q should also not be close to one another. If p and q are close to one another,

(p-q)/ 2 will be small.

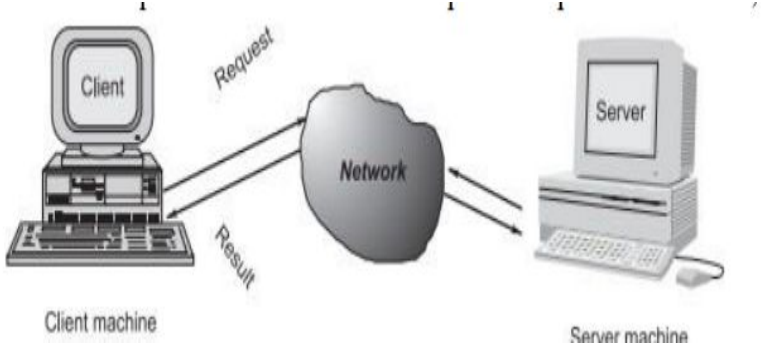
(p+q)/2 will be only slightly larger than $\sqrt{n}$

$(p+q)2 /4 - n = (p-q)2 / 4$.

So to factorize n, keep checking integers $x > \sqrt{n}$ such than x2 - n is a perfect square, say y2 . Then p = x + y and q = x - y.

Example: n = 97343 $\sqrt{n}$ = 311: So p = x + y = 313 and q = x - y = 311. For this reason, it is advisable that p and q are such that their bit representations differ in length by a few bits. The two primes, p and q, which compose the modulus, should be of roughly equal length; this will make the modulus harder to factor than if one of the primes was very small. Thus if one chooses to use a 768-bit modulus, the primes should each have length approximately 384 bits. If the two primes are extremely close (identical except for, say, 100 - 200 bits), there is a potential security risk, but the probability that two randomly chosen primes are so close is negligible. 8 Note: Every RSA cryptosystem has some plain-text blocks which are encrypted into itself (in fact, at least four such blocks). For instance, 1,21,34,54 are plain-texts which are encrypted into itself for the first example.

Socket Programming:



Generally, programs running on client machines make requests to a program (often called as server program) running on a server machine. They involve networking services provided by the transport layer, which is part of the Internet software stack, often called TCP/IP (Transport Control Protocol/Internet Protocol) stack. The transport layer comprises two types of protocols, TCP (Transport Control Protocol) and UDP (User Datagram Protocol). The most widely used programming interfaces for these protocols are sockets. TCP is a connection-oriented protocol that provides a reliable fl ow of data between two computers. Example applications that use such services are HTTP, FTP, and Telnet. UDP is a protocol that sends independent packets of data, calleddatagrams, from one computer to another with no guarantees about arrival and sequencing. Example applications that use such services include Clock server and Ping. The TCP and UDP protocols use ports to map incoming data to a particular process running on a computer.

Sockets provide an interface for programming networks at the transport layer. Network communication using Sockets is very much similar to performing file I/O. In fact, socket handle is treated like file handle. The streams used in file I/O operation are also applicable to socket-based I/O. Socket-based communication is independent of a programming language used for implementing it. A server (program) runs on a specific computer and has a socket that is bound to a specific port. The server listens to the socket for a client to make a connection request. If everything goes well, the server accepts the connection. Upon acceptance, the server gets a new socket bound to a different port. It needs a new socket (consequently a different port number) so that it can continue to listen to the original socket for connection requests while serving the connected client.

Procedural Description:

Transmitting a message between Client and Server UDP Application using python Programming.

Plaintext will be converted into cipher text using RSA Algorithm.

Server has been developed using Server Socket programming in python

Client has been developed using Socket Programming in python

The program runs on the any system irrespective of the OS. But it requires any python console.

We convert the cipher text into a cipher text which is suitable for transmission through socket programming. Various inbuilt commands are used to implement RSA Algorithm and socket programming simultaneously

Operations involved on Server side code:

This is the procedure happening at the server side:-

Creates a socket to make a connection with client

Waits for the receiving the public key from client to encrypt the message.

Encrypts the message using received keys using socket.

Changing encrypted cipher text into cipher text which is suitable for transmitting using socket programming.

Send its public key to client to encrypt message.

Receive encrypted message and decrypt it using private keys.

Repeating the process till exit is not executed.

Operations involved on Client side code:

This is the procedure happening at the client side:-

Creates a socket to make a connection with server.

Sending the public key from client to encrypt the message. Keys are changed into a number suitable for transmitting using socket programming.

Waits for the Encrypted message to receive.

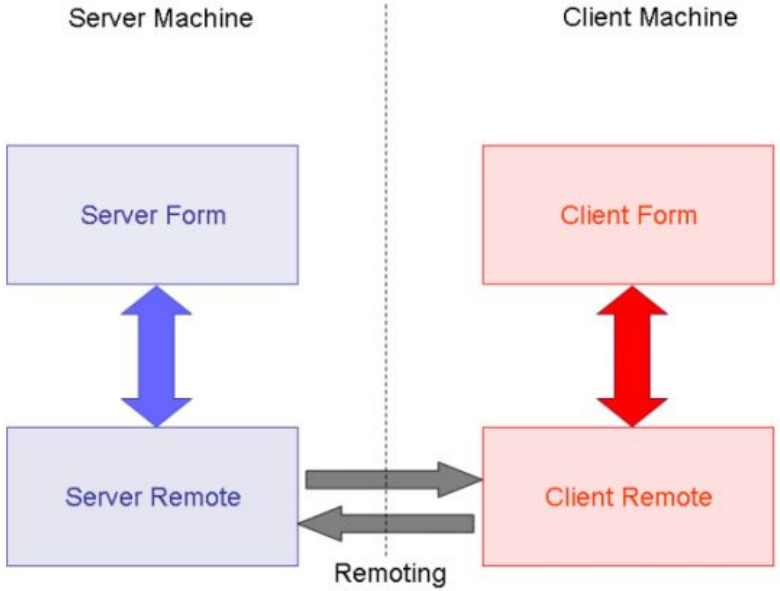
Decrypt the received cipher text using private keys.

Waits for receiving the public key from server to encrypt the message.

Encrypt the message using received public key.

Repeating the process till exit is not executed.

Architecture Diagram:

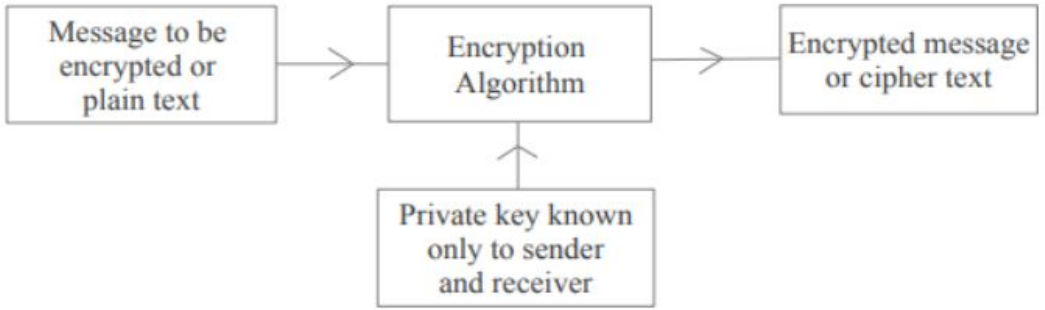


IMPLEMENTATION WORK

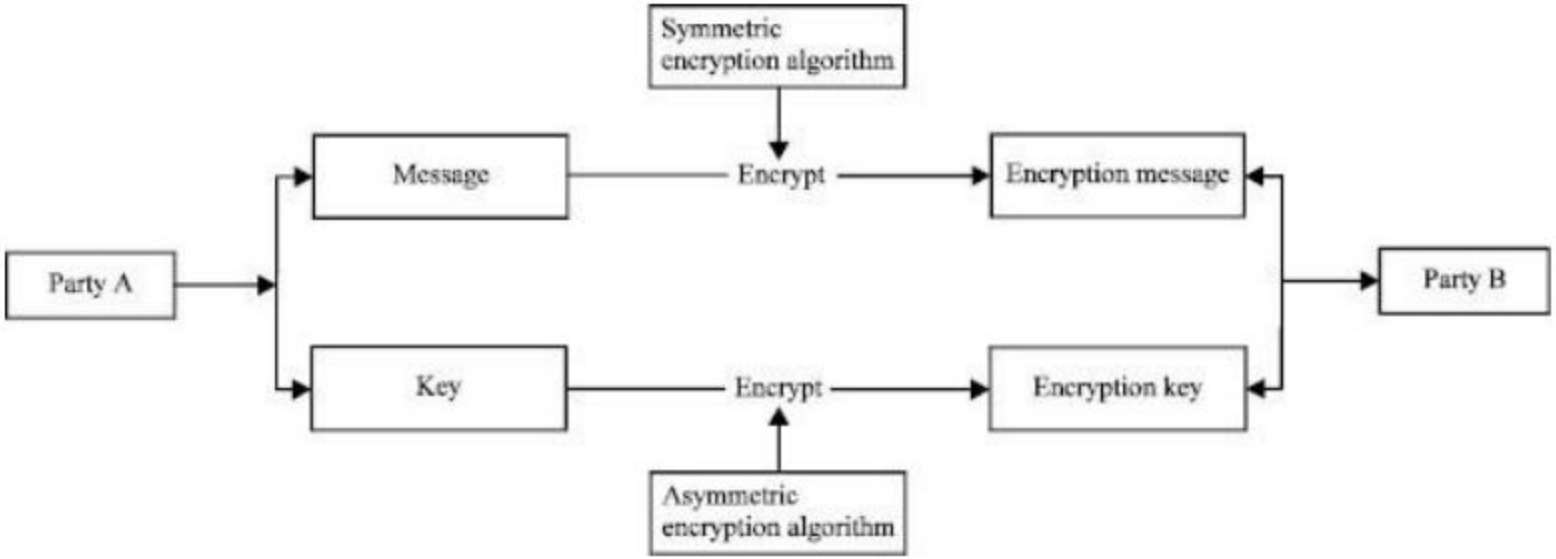
Below is the flow diagram of how we are going to implement our work.

Diagrams

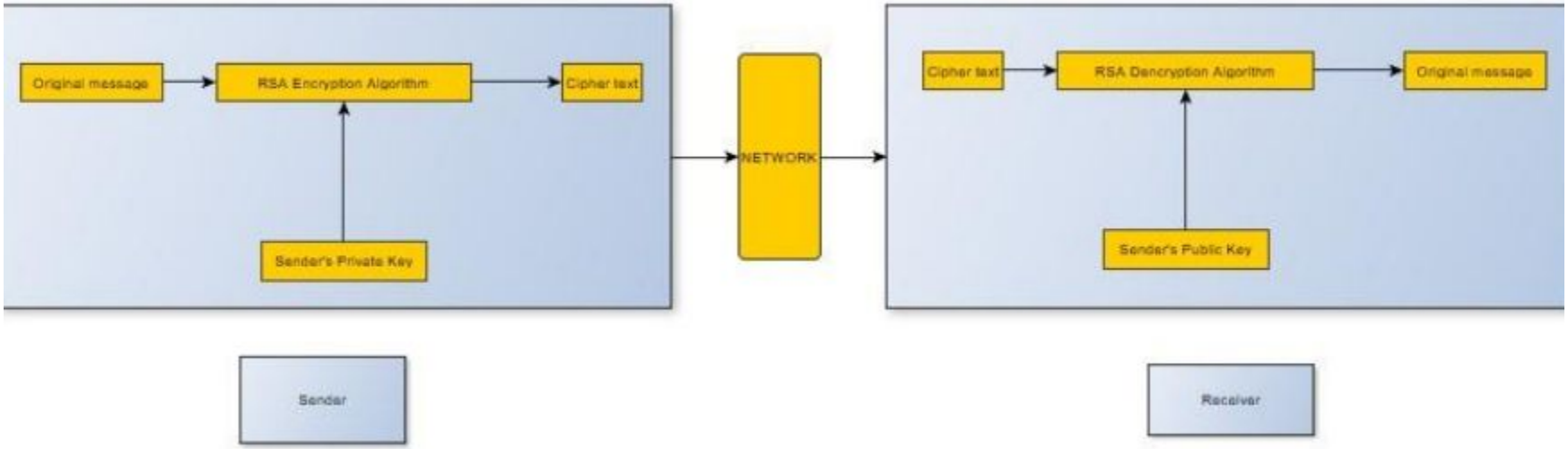
Data F



Basic Flow diagram for Data Security using Encryption Decryption process:


Symmetric encryption algorithm
Message
Encrypt
Encryption message
Party A
Party B
Key
Encrypt
Encryption key
Asymmetric encryption algorithm

Procedural flow diagram of the code:



Innovation

We have used RSA algorithm for the chat application. Here the plaintext of the chat is also not stored in the server this way even though the intruder has entered into the server, he won’t be able to access the data. We have also built the GUI for this chat application for the ease access of the user.

CODE:

Server side code:

```
import random
"""Server for multithreaded (asynchronous) chat application."""
from socket import AF_INET, socket, SOCK_STREAM
from threading import Thread
def passwordcheck(n,pas):
    if n=="koushik " and pas=="1234":
        return True
    elif n=="sahithi " and pas=="sahi":
        return True
    return False
def accept_incoming_connections():
    """Sets up handling for incoming clients."""
```

```
while True:
    client, client_address = SERVER.accept()
    print("%s:%s has connected." % client_address)
    client.send(bytes("Greetings from the cave! Now type your user id and press enter!"))
    l=[]
    l.append(client_address)
    addresses[client] = l
    Thread(target=handle_client, args=(client,)).start()
```

```
def handle_client(client):  # Takes client socket as argument.
    """Handles a single client connection."""
    name = client.recv(BUFSIZ).decode("utf8")
    addresses[client].append(name)
    welcome = 'Welcome %s! If you ever want to quit, type {quit} to exit.' % name
    client.send(bytes(welcome))
    client.send(bytes("Enter the password"))
    pas = client.recv(BUFSIZ).decode("utf8")
    if(passwordcheck(name,pas)):
        client.send(bytes("Enter the user id of your friend"))
        n1 = client.recv(BUFSIZ).decode("utf8")
        client.send(bytes("U can chat with ur frn "+str(n1)))
        print(n1)
        clients[client] = name
        while True:
            msg = client.recv(BUFSIZ)
            if msg != bytes("{quit}"):
                print(str(addresses[client])+" "+msg)
                broadcast(msg, name)
```

```
else:
    client.send(bytes("{quit}"))
    client.close()
    del clients[client]
    broadcast(bytes("This person has left the chat: ",name))
    break4
```

```
    else:
        wr = 'invalid password. restart the session to start again'
        client.send(bytes(wr))
def broadcast(msg, n1=""):
    # prefix is for name identification.
    """Broadcasts a message to all the clients."""
    for sock in clients:
        print(n1)
  if addresses[sock][1]==n1:
            #m=prefix+str(n1)+" "+msg
            m=msg+'-'+n1
            #print(m,addresses[sock][1],1)
            sock.send(bytes(m))
   else:
            #m=prefix+str(n1)+" "+msg
            m=msg+'-'+n1
            print(m,addresses[sock][1],2)
            sock.send(bytes(m))
```

```
clients = {}

addresses = {}

HOST = ''

PORT = random.randint(100,10000)
```

```
BUFSIZ = 1024
ADDR = (HOST, PORT)
SERVER = socket(AF_INET, SOCK_STREAM)
SERVER.bind(ADDR)
if __name__ == "__main__":
    SERVER.listen(5)
    print("Waiting for connection... at "+str(PORT))
    ACCEPT_THREAD = Thread(target=accept_incoming_connections)
    ACCEPT_THREAD.start()
    ACCEPT_THREAD.join()
    SERVER.close()
```

Client side code:

```
from socket import AF_INET, socket, SOCK_STREAM
from threading import Thread
from Tkinter import *
import tkFont
#import tkinter
import csv
cid=0
count=0
cl=0
n,d=0,0
lastmsg=""
lastciph=""
uid=""
```

```
def user(id1):
    with open('keys.csv', 'r') as csvfile:
        spamreader = csv.reader(csvfile, delimiter=' ')
        global cl,n,d,uid
        for row in spamreader:
                r=row[0].split(',')
                print(r[0],id1)
            cl=cl+1
                if r[0]==id1:
                    uid=id1
                    f1=open('pr'+str(cl)+'.txt', 'r')
                    n,d=map(long,f1.read().split())
    return False
```

```
def keyext(id1):
    with open('keys.csv', 'r') as csvfile:
        spamreader = csv.reader(csvfile, delimiter=' ')
        for row in spamreader:
                r=row[0].split(',')
                print(r[0],id1)
                if r[0]==id1:
                    return r[1:]
    return False
def decrypt(ciplat,sp):
    global d,n
    ciplat=ciplat.encode('ascii')
    sp=int(sp.encode('ascii'))
    nl=len(str(n))
```

```
    actual_msg=""
    for i in range(0,len(ciplat),sp):
        np=(ciplat[i:i+sp])
          print(np)
          if np.lstrip('0')==":
            actual_msg=actual_msg+'0'
          else:
            print(np,d,n)
            no=(long(np)**d)%n
            print(no)
            actual_msg=actual_msg+str(no)
        print(actual_msg)
num_msg=bin(long(actual_msg))[3:]
    print('Actual msg in binary -'+num_msg)
    MESSAGE=""
    for j in range(8,len(num_msg)+1,8):
            t=num_msg[j-8:j]
            t1=int(t,2)/2
```

```
            if t1==64:
                t1=t1/2
            print(t1,chr(t1))
            MESSAGE=MESSAGE+chr(t1)
    print(MESSAGE)
    return MESSAGE
def encrypt(msg,k):
    global lastmsg,lastciph
    c=""
    n2=long(k[1])
```

```
    e2=long(k[0])
    print('e2,n2',e2,n2)
    for i in msg:
b=bin(ord(i))
        b=b[2:]
        if len(b)<8:
        b=b+('0'*(8-len(b)))
        print(i,b)
        c=c+b
    print(c)
    num_msg=int('1'+c,2)
    print(num_msg)
cipher=[]
    sp=len(str(n2))-1
    num=str(num_msg)
    for i in range(0,len(num),sp):
        par=num[i:i+sp]
        print(par,type(par))
```

```
    while(par[0]=='0'):
        cipher.append(0)
        par=par[1:]
    par=long(par)
    cipher.append((par**e2)%n2)
    print('Cipher part '+str(cipher[-1]))
le=len(str(max(cipher)))
cipherfin=""
for ci in cipher:
    if len(str(ci))<le:
```

```
            add=le-len(str(ci))
            cipherfin=cipherfin+('0'*add)+str(ci)
        else:
            cipherfin=cipherfin+str(ci)
        print(ci,cipherfin)
    print("ciphertext final= "+str(cipherfin))
    cipherfin=cipherfin+'-'+str(le)
    lastmsg,lastciph=msg,cipherfin
    return cipherfin
```

```
def receive():
    """Handles receiving of messages."""
    while True:
     try:
            global lastmsg,lastciph
            msg = client_socket.recv(BUFSIZ).decode("utf8")


    if count>1 and cid!=0:
                #msg.config(font=norm_font, foreground="yellow",background="lightblue")
                print(msg)
                f=msg.rfind('-')
                m1=msg[0:f]
                print(m1)
                if m1==lastciph:
                   print('cas1',lastmsg,lastciph)
                   msg_list.insert(END, uid+': '+lastmsg)
                else:
```

```
            print('cas2')
            ci,sp,id2=msg.split('-')
            msg.replace('(',")
            msg.replace(')',")
            #sp=msg[sto:].replace(',')
           print(ci,sp,id2)
            msg=decrypt(ci,sp)
            print(msg)
            msg_list.insert(END, id2+': '+msg)
        else:
        #msg.config(font=norm_font, foreground="yellow",background="lightblue")
        print(msg)
        msg_list.insert(END, msg)
except OSError:  # Possibly client has left the chat.
    Break
```

```
def send(event=None):  # event is passed by binders.
    """Handles sending of messages."""
    msg = my_msg.get()
    print(msg)
    my_msg.set("")  # Clears input field.
    global cid
    global count
    if count>1 and cid!=0:
        msg=encrypt(msg,cid)
        client_socket.send(bytes(msg))
    else:
        client_socket.send(bytes(msg))
    if msg == "{quit}":
```

```
        client_socket.close()
        top.quit()
    if count==0:
        user(msg)
          if count==1:
        k=keyext(msg)
        print(k)
        cid=k
    count=count+1
```

```
def on_closing(event=None):
    """This function is to be called when the window is closed."""
    my_msg.set("{quit}")
    send()
top = Tk()
top.title("WLAN CHAT")
messages_frame = Frame(top)
my_msg = StringVar()  # For the messages to be sent.
norm_font = tkFont.Font(family="Calibri",size=18)
#my_msg.tag_config("hyper",font=norm_font, foreground="white", underline=1)
my_msg.set("")
scrollbar = Scrollbar(messages_frame)  # To navigate through past messages.
# Following will contain the messages.
msg_list = Listbox(messages_frame, height=20, width=50, yscrollcommand=scrollbar.set)
msg_list.config(font=norm_font, foreground="red",background="lightblue")
scrollbar.pack(side=RIGHT, fill=Y)
msg_list.pack(side=LEFT, fill=BOTH)
```

```
msg_list.pack()
messages_frame.pack()
entry_field = Entry(top, textvariable=my_msg)
entry_field.config(font=norm_font, foreground="red")
entry_field.bind("<Return>", send)
entry_field.pack()
send_button = Button(top, text="Send", command=send)
send_button.pack()
top.protocol("WM_DELETE_WINDOW", on_closing)
#----Now comes the sockets part----

HOST = '192.168.43.94'
PORT = raw_input('Enter port: ')
if not PORT:
    PORT = 33000
else:
    PORT = int(PORT)
BUFSIZ = 1024
ADDR = (HOST, PORT)
client_socket = socket(AF_INET, SOCK_STREAM)
client_socket.connect(ADDR)
receive_thread = Thread(target=receive)
receive_thread.start()
mainloop()  # Starts GUI execution.
```

RSA key generator

```
import random
from fractions import gcd
def egcd(a, b):
    x, lastX = 0, 1
    y, lastY = 1, 0
    while (b != 0):
        q = a // b
        a, b = b, a % b
        x, lastX = lastX - q * x, x
        y, lastY = lastY - q * y, y
    print(x,lastX,y,lastY)
    return lastY

def checkprime(n):
    for i in (2, n):
        if n%i==0:
            return 0
    return 1
def rsa_gen():
    print("enter range for prime numbers")
    x=int(input())
    y=int(input())
```

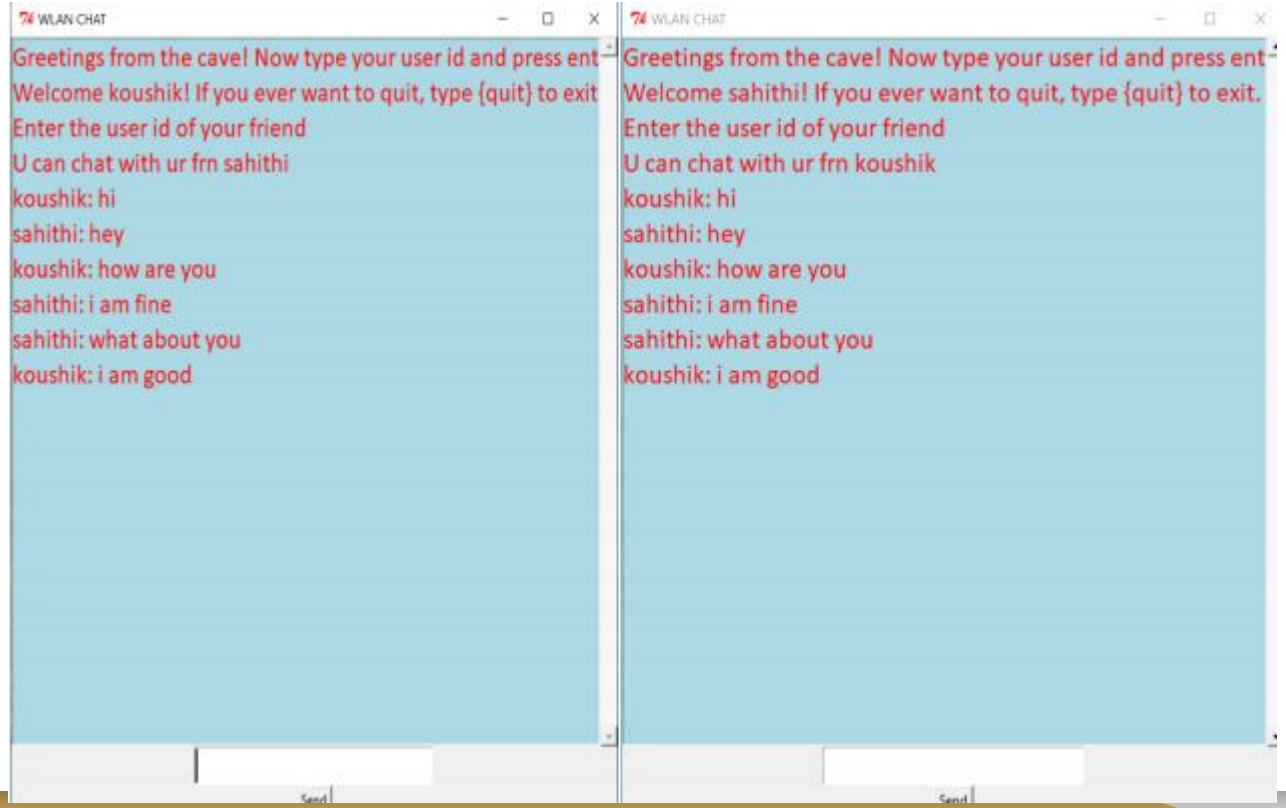
```
pr=[]
for num in range(x,y+1):
  # prime numbers are greater than 1
  if num > 1:
     for i in range(2,num):
```

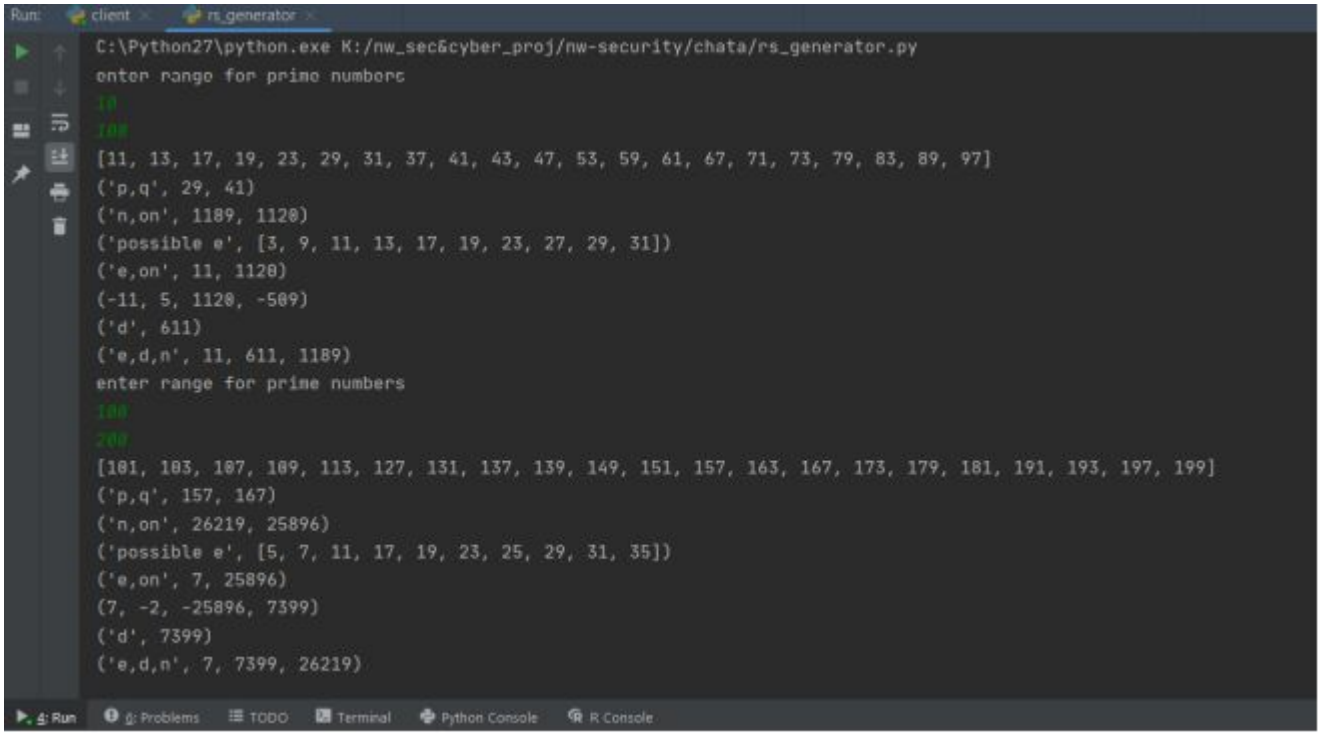
```
            if (num % i) == 0:
                break
            if i==num-1:
                pr.append(num)
print(pr)
p=random.choice(pr)
q=random.choice(pr)
while(p==q):
    q=random.choice(pr)
print("p,q",p,q)
n=p*q
on=(p-1)*(q-1)
print("n,on",n,on)
l=[] # possible e values
```

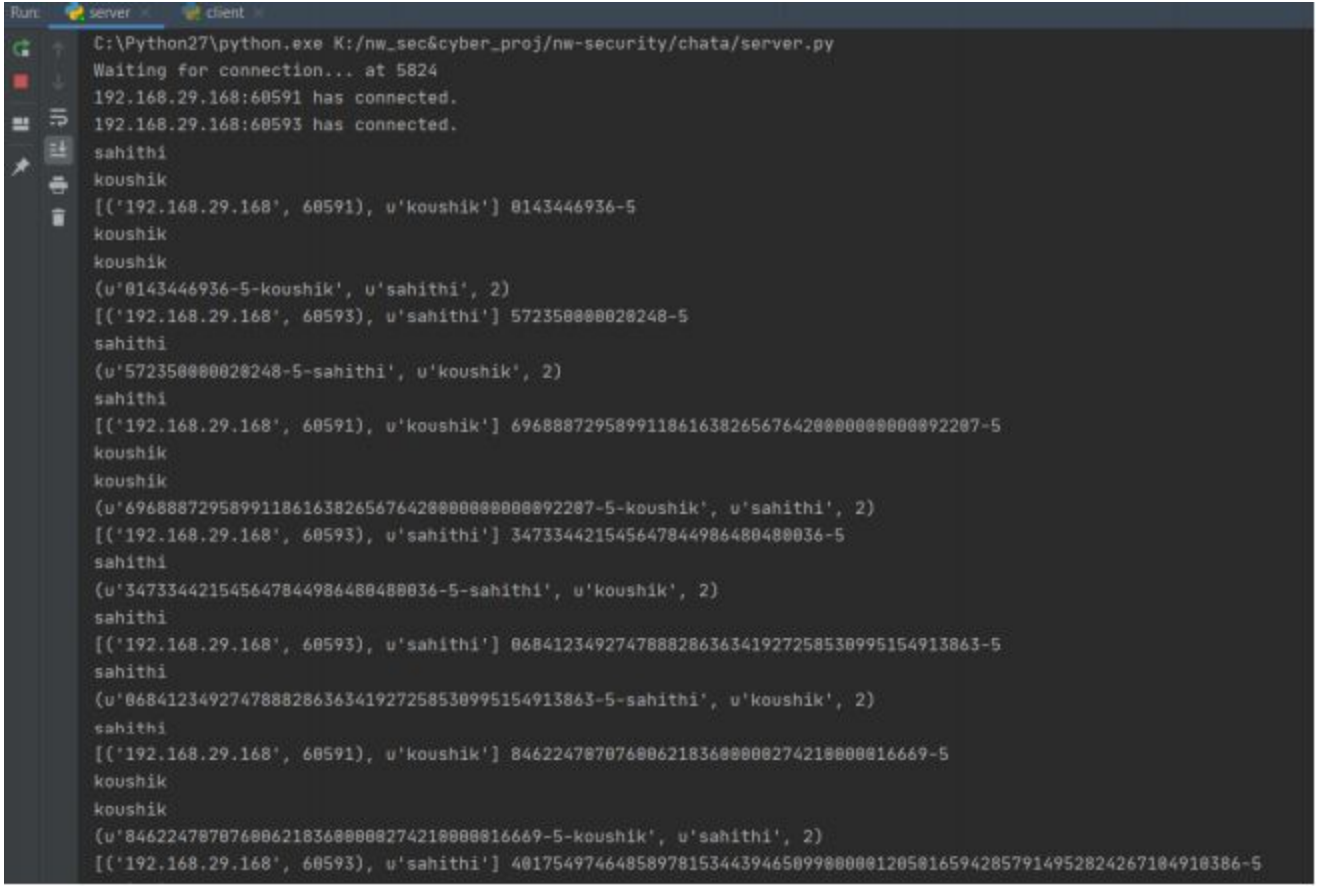
```
    for i in range(2,on):
        if gcd(i,on)==1:
            l.append(i)
        if len(l)==10:
            break
    print('possible e', l)
    e=random.choice(l)
    print("e,on",e,on)
    d=egcd(on,e)
    if d<0:
        d=d+on
    print("d",d)
    return e,d,n
e1,d1,n1=rsa_gen()
```

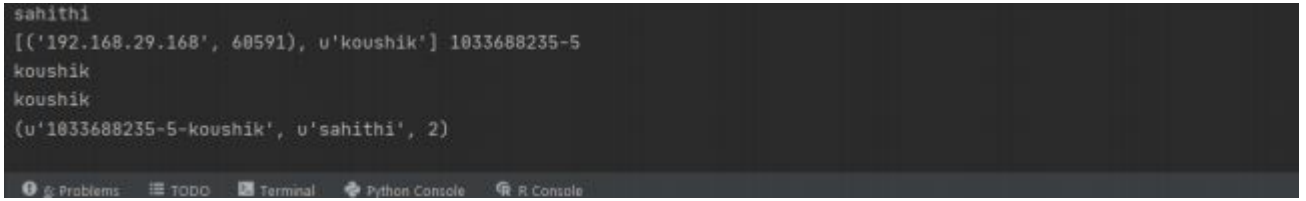
```
print('e,d,n',e1,d1,n1)

e2,d2,n2=rsa_gen()

print('e,d,n',e2,d2,n2)
```

OUTPUT:


WLAN CHAT
Greetings from the cave! Now type your user id and press ent
Welcome koushik! If you ever want to quit, type {quit} to exit
Enter the user id of your friend
U can chat with ur frn sahithi
koushik: hi
sahithi: hey
koushik: how are you
sahithi: i am fine
sahithi: what about you
koushik: i am good
Send
WLAN CHAT
Greetings from the cave! Now type your user id and press ent
Welcome sahithi! If you ever want to quit, type {quit} to exit.
Enter the user id of your friend
U can chat with ur frn koushik
koushik: hi
sahithi: hey
koushik: how are you
sahithi: i am fine
sahithi: what about you
koushik: i am good
Send


Run:
client
rs_generator
C:\Python27\python.exe K:/nw_sec&cyber_proj/nw-security/chata/rs_generator.py
enter range for prime numbers
10
100
[11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]
('p,q', 29, 41)
('n,on', 1189, 1120)
('possible e', [3, 9, 11, 13, 17, 19, 23, 27, 29, 31])
('e,on', 11, 1120)
(-11, 5, 1120, -509)
('d', 611)
('e,d,n', 11, 611, 1189)
enter range for prime numbers
100
200
[101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199]
('p,q', 157, 167)
('n,on', 26219, 25896)
('possible e', [5, 7, 11, 17, 19, 23, 25, 29, 31, 35])
('e,on', 7, 25896)
(7, -2, -25896, 7399)
('d', 7399)
('e,d,n', 7, 7399, 26219)
4: Run
6: Problems
TODO
Terminal
Python Console
R Console


Run:
server
client
C:\Python27\python.exe K:/nw_sec&cyber_proj/nw-security/chata/server.py
Waiting for connection... at 5824
192.168.29.168:60591 has connected.
192.168.29.168:60593 has connected.
sahithi
koushik
[('192.168.29.168', 60591), u'koushik'] 0143446936-5
koushik
koushik
(u'0143446936-5-koushik', u'sahithi', 2)
[('192.168.29.168', 60593), u'sahithi'] 572350800028248-5
sahithi
(u'572350800028248-5-sahithi', u'koushik', 2)
sahithi
[('192.168.29.168', 60591), u'koushik'] 6968887295899118616382656764280000000000892207-5
koushik
koushik
(u'6968887295899118616382656764280000000000892207-5-koushik', u'sahithi', 2)
[('192.168.29.168', 60593), u'sahithi'] 34733442154564784449864804800 36-5
sahithi
(u'34733442154564784449864804800 36-5-sahithi', u'koushik', 2)
sahithi
[('192.168.29.168', 60593), u'sahithi'] 0684123492747888286363419272585309951549138 63-5
sahithi
(u'0684123492747888286363419272585309951549138 63-5-sahithi', u'koushik', 2)
sahithi
[('192.168.29.168', 60591), u'koushik'] 8462247807076806218368000027421800081 6669-5
koushik
koushik
(u'8462247807076806218368000027421800081 6669-5-koushik', u'sahithi', 2)
[('192.168.29.168', 60593), u'sahithi'] 401754974648589781534439465099800001205816594285791495282426710491 0386-5



TEST CASES:

Case No.	Port	Friend	Password	Output
Test Case1	Wrong port number given	X	X	Invalid port number
Test Case 2	Correct port	Wrong friend	X	Unable to find friend "…..
Test Case 3	Correct port	Correct friend	Wrong Password	Invalid pass word ente again
Test Case 4	Correct port	Correct friend	Correct Password	Successfully Connected

CONCLUSIONS

We have studied the RSA public-key cryptosystem. We have delved into the basis for its working, its strength, and its ease of understanding and use. We have dealt mainly from the point of view of legal users. RSA is the most popular public-key cryptosystem available today. Its popularity stems from the fact that it can be used for both encryption and authentication, and that it has been around for many years and has successfully withstood much scrutiny. The security of RSA is related to the assumption that factoring is difficult. An easy factoring method or some other feasible attack would break RSA.RSA is built into current operating systems by Microsoft, Apple, Sun, and Novell. In hardware, RSA can be found in secure telephones, on Ethernet network cards, and on smart cards. In addition, RSA is incorporated into all of the major protocols for secure Internet communications. The estimated installed base of RSA encryption engines is around 20 million, making it by far the most widely used public-key cryptosystem in the world

REFERENCES

[1] Afolabi, A.O and E.R. Adagunodo, 2012. Implementation of an Improved data encryption algorithm in a web based learning system. International Journal of research and reviews in Computer Science. Vol. 3, No. 1. [2] Bhoopendra, S.R., Prashanna, G. and S. Yadav, 2010. An Integrated encryption scheme used in Bluetooth communication mechanism. International Journal of Computer Tech. and Electronics Engineering (IJCTEE), vol. 1, issue 2.

[3] DI management (2005) “RSA algorithm”, available at: http://www.dimgt.com.au/rsa_alg.html.

[4] Gaurav, S., 2012. Secure file transmission scheme based On hybrid encryption technique. International Journal of management, IT and Engineering. Vol. 2, issue 1.

[5] Hellman, M. and J. Diffie, 1976. New Directions in Cryptography. IEEE transactions on Information theory, vol. IT-22, pp:644-654.

[6] Shinde, G.N. and H.S. Fade War, 2008. Faster RSA algorithm for decryption using Chinese remainder theorem. ICCES, Vol. 5, No. 4, pp. 255-261.
[7] Yang L. and S.H. Yang. 2007. A frame work of security and safety checking for internet-based control systems. International Journal of Information and Computer security. Vol.1, No. 2.
[8] Washington, L.C. 2006. Introduction to Cryptography: with coding theory by Wade Trappe. Upper Saddle River, New Jersey, Pearson Prentice Hall.
[9] Wuling Ren College of Computer and Information Engineering Zhejiang Gongshang University. 2010. A hybrid encryption algorithm based on DES and RSA in Bluetooth communication. Second International Conference on Modeling, Simulation and Visualization methods.
[10] RSA Security Inc., “PKCS #1 v2.0 amendment 1: Multi-prime RSA,” July 2000. Available ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-0a1.pdf.