AMRITA VISHWAVIDYAPEETHAM

COIMBATORE, ETTIMADAI

APPLIED CRYPTOGRAPHY

21AIE431

COLEMAK SUBSTITUTION CIPHER (TWO LAYERED ENCRYPTION METHODOLOGY)

DEPARTMENT	**-**	**AIE-CEN**
COURSE	**-**	**APPLIED CRYPTOGRAPHY**
SEMESTER	**-**	**5**
INSTRUCTOR	**-**	**SUNIL KUMAR S**

GROUP – 3 BATCH-A

GROUP MEMBERS:

S.NO	NAME	ROLL NO
1	P.VENU.S.G.V. KIRAN	CB.EN.U4AIE21044
2	VENKATA SIVA MANOJ ADDALA	CB.EN.U4AIE21074
3	VELAGA LELA LAKHMAN	CB.EN.U4AIE21073
4	KAMUJU AKASH	CB.EN.U4AIE21021

ACKNOWLEDGEMENT

We would like to express our special thanks of gratitude to our teacher (SUNIL KUMAR S sir), who gave us the golden opportunity to do this wonderful project on the topic (COLEMAK SUBSTITUTION CIPHER (TWO LAYERED ENCRYPTION METHODOLOGY)), which also helped us in doing a lot of Research and we came to know about so many new things. We are thankful for the opportunity given.We would also like to thank our group members, as without their co-operation, we would not have been able to complete the project within the prescribed time.

DECLARATION

We declare that the Submitted Report is our original work and no values and context of it has been copy pasted from anywhere else. We take full responsibility, that if in future, the report is found invalid or copied, the last decision will be of the Faculty concerned. Any form of plagiarism will lead to disqualification of the report.

TABLE OF CONTENTS

ABSTRACT

The chat application's client-server architecture combines asymmetric (RSA) and symmetric (COLEMAK) encryption techniques for secure message exchange. RSA encryption involves generating public and private keys for secure communication, while COLEMAK adds an extra layer of security through substitution ciphers with a unique keyboard-inspired alphabet. Message forwarding includes two encryption layers: RSA encryption with the recipient's public key, followed by COLEMAK encryption with a global public key (colekey). Decryption involves reversing this process, first using COLEMAK and then RSA with the recipient's private key. The application establishes secure connections over TCP and uses the colekey, derived from device-specific information, to enhance security. The chat application is executed within a user-friendly GUI, ensuring confidentiality and robustness in communication.

INTRODUCTION

This application is combination of both symmetric and asymmetric techniques.

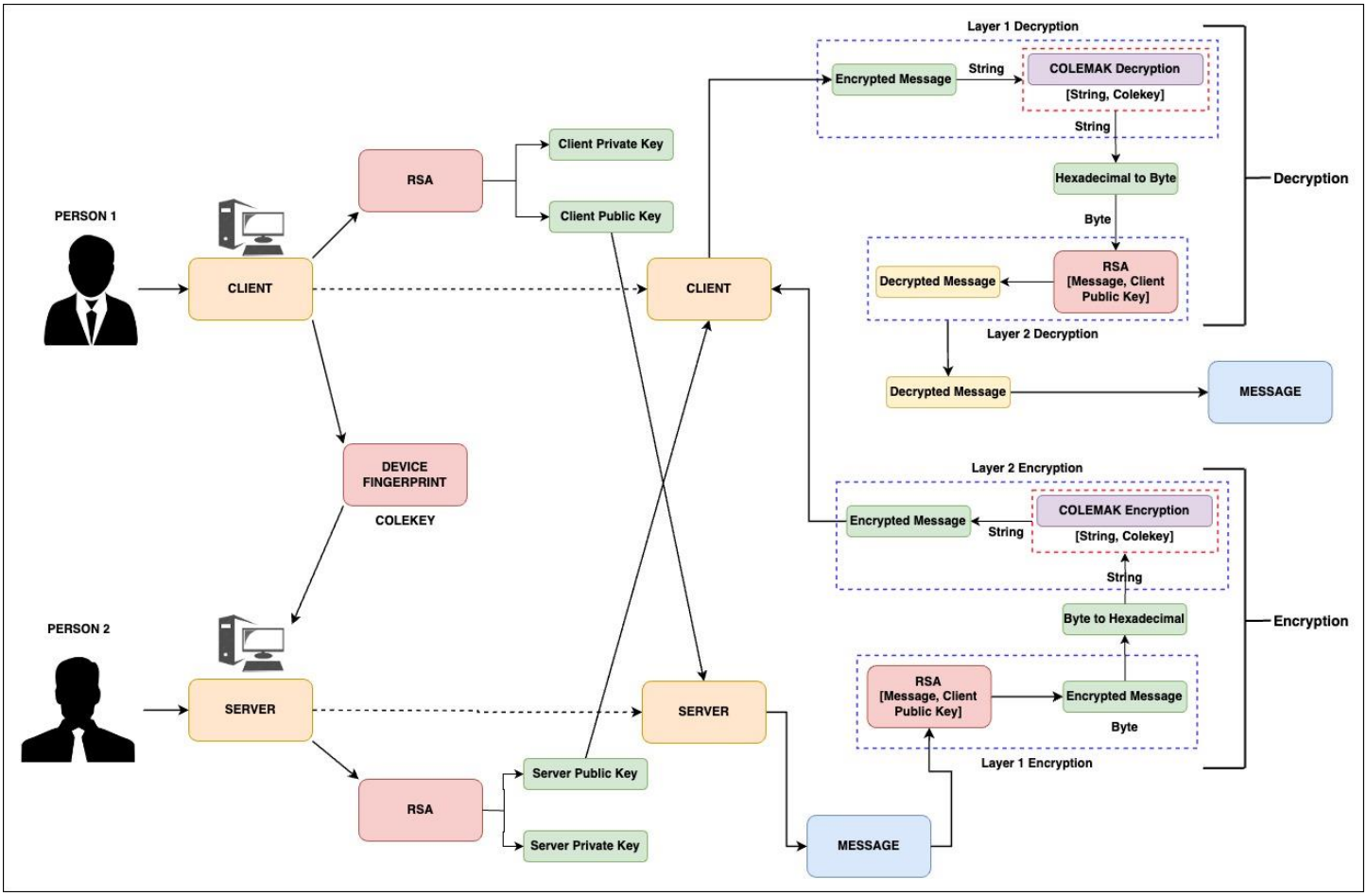
Encryption and decryption of messages play a pivotal role in ensuring the confidentiality, integrity, and privacy of communication in today's digital landscape. By encrypting messages, sensitive information is transformed into unreadable ciphertext that can only be deciphered by authorized recipients holding the decryption keys. This safeguards data from unauthorized access and cyber threats, providing a secure means for businesses, individuals, and governments to protect their digital assets, maintain privacy, and uphold trust in the online exchange of information. In an era marked by increasing digital risks and vulnerabilities, encryption and decryption are indispensable tools for safeguarding the confidentiality and integrity of sensitive data.

RSA(asymmetric) encryption involves generating a key pair, consisting of a public key for encryption and a private key for decryption. The sender encrypts a message using the recipient's public key, and the recipient subsequently decrypts the ciphertext using their private key to retrieve the original message.

In parallel, the application enhances security through COLEMAK(symmetric), a traditional rotating cipher method with a unique initial set of alphabets inspired by the relatively obscure COLEMAK keyboard layout. This layout, being unfamiliar to most, adds an extra layer of unpredictability. COLEMAK is employed as the basis set for substitution cipher encryption, which is further fortified by incorporating a device fingerprint (colekey). This combination makes decryption significantly more challenging without access to the specific key.

The process of message forwarding involves two successive layers of encryption. Firstly, the sender encrypts the message using the recipient's public key via RSA encryption, constituting the first layer of security. Subsequently, the already encrypted message is further protected by applying COLEMAK substitution cipher with the global public key (colekey). Upon receiving the message, the recipient performs the decryption process in reverse order. First, they decrypt the message using COLEMAK, serving as the first layer of decryption. Then, the second layer of decryption employs the recipient's private key to recover the original message. This two-layer encryption scheme ensures a high level of message security and confidentiality.

METHODOLOGY:



The communication process initiates with a TCP connection establishment between the client and server. Subsequently, the RSA algorithm is employed to generate public and private keys for both the client and server. These generated public keys are then securely exchanged between the client and server to establish a foundation for secure communication.

A pivotal element in enhancing security is the "colekey," a global public key utilized in the COLEMAK substitution cipher. This colekey is created by combining various device-specific information such as the MAC address, OS details, hardware specifications, network parameters, and UUID, effectively constituting a unique device fingerprint.

When a sender transmits a message (denoted as "E"), it undergoes a two-layer encryption process to ensure robust security. Firstly, the message is encrypted as "E1" using RSA encryption with the recipient's public key. This initial encryption results in "E1" being in byte format. To facilitate further processing, this byte format is converted to a hexadecimal string denoted as "H1." Subsequently, "H1" is subjected to a second layer of

encryption utilizing the COLEMAK substitution cipher with the colekey as the global public key. The outcome is "E2," representing the fully encrypted message, which is then forwarded to the recipient.

Upon reception, "E2" is subjected to a decryption process in two sequential layers. Initially, "D1" is derived by decrypting "E2" using the COLEMAK substitution cipher with the colekey. The resulting "D1" is in string format. To facilitate the RSA decryption process, it is converted back to bytes using the "hextobyte()" function. This yields "H2" in byte format. Finally, "H2" is decrypted in the second layer using RSA encryption with the recipient's private key, ultimately yielding the original message, "D2," as the final decrypted content.

This chat application is executed within a graphical user interface (GUI), providing an intuitive and user-friendly interface for secure and seamless communication.

CODES:

COLEMAK SUBSTITUTION CIPHER – DEVICE FINGERPRINTING:

```
def generate_device_fingerprint():
    if os.path.exists("device_uuid.txt"):
        with open("device_uuid.txt", "r") as file:
            uuid_str = file.read().strip()
    else:
        uuid_str = str(uuid.uuid4())
        with open("device_uuid.txt", "w") as file:
            file.write(uuid_str)

    fingerprint_data = {
        "MAC add" : get_mac_address(),
        "os_info": get_os_info(),
        "hardware_info": get_hardware_info(),
        "network_info": get_network_info(),
        "uuid": uuid_str
    }
```

```
    fingerprint_json = json.dumps(fingerprint_data, sort_keys=True)
    fingerprint_hash = hashlib.md5(fingerprint_json.encode()).hexdigest()

    return fingerprint_hash
```

COLEMAK SUBSTITUTION CIPHER CODE:

```
import test as secret

char_set = ['q', 'w', 'f', 'p', 'g', 'j', 'l', 'u', 'y', 'a', 'r', 's', 't',
            'd', 'h', 'n', 'e', 'i', 'o', 'z', 'x', 'c', 'v', 'b', 'k', 'm',
            '1', '2', '3', '4', '5', '6', '7', '8', '9', '0',
            '!', '@', '#', '$', '%', '^', '&', '*', '(', ')']
```

CALMAK SUBSTITUTION CIPHER ENCRYPTION:

```
def encrypt(text, key) :

    crypt = ''
    l1 = len(text)
    l2 = len(key)
    key = key * (int(l1/l2) + (l1%l2)) + key[0 : (l1%l2)]

    for i in range (len(text)) :
        if text[i] == ' ' :
            crypt = crypt + " "

        elif text[i].lower() in char_set :
            num = char_set.index(key[i].lower())
            index = char_set.index(text[i].lower())
            if num != 0 :
                crypt = crypt + char_set[(index + num) % len(char_set)]
            else :
                crypt = crypt + char_set[(index + 4) % len(char_set)]
```

```
    elif text[i] not in char_set :
        crypt = crypt + text[i]
         crypt = decase(crypt, case(text))

  return crypt
```

COLEMAK SUBSTITUTION CIPHER DECRYPTION :

```
def decrypt(crypt, key):

  text = ''
  l1 = len(crypt)
  l2 = len(key)
  key = key * (int(l1/l2) + (l1%l2)) + key[0 : (l1%l2)]

  for i in range (len(crypt)) :
    if crypt[i] == ' ' :
      text = text + " "

    elif crypt[i].lower() in char_set :
      num = - char_set.index(key[i].lower())
      index = char_set.index(crypt[i].lower())
      if num != 0 :
        text = text + char_set[(index + num) % len(char_set)]
      else :
        text = text + char_set[(index - 4) % len(char_set)]
    elif crypt[i] not in char_set :
      text = text + crypt[i]

  text = decase(text, case(crypt))

  return text
```

COLEMAK SUBSTITUTION CIPHER CASING AND DECASING:

```
def case (text) :
   bin = ""
   for i in text :
      if i == " " :
         bin = bin + " "
      elif i.isupper() :
         bin = bin + "1"
      else :
         bin = bin + "0"
   return bin


def decase (text, bin) :
   out = ""
   for i in range (len(text)) :
      if bin[i] == " " :
         out = out + " "
      elif bin[i] == "1" :
         out = out + text[i].upper()
      else :
         out = out + text[i].lower()
   return out
```

COLEMAK SUBSTITUTION CIPHER IMPROVISED:

```
import rsa
import COLEMAK as colemak
import binascii

def genkey():
   publicKey, privateKey = rsa.newkeys(512)
   return publicKey, privateKey
```

```
publicKey, privateKey = genkey()
def enc(message, key, colekey):
    encMessage = rsa.encrypt(message.encode(), key)
    print("-------- LEVEL 1 ENCRYPTION : ---------")
    print(encMessage)
    print(" ")
    print("-------- LEVEL 2 ENCRYPTION : ---------")
    encMessage_colemak = cole_enc(encMessage, colekey)
    print(encMessage_colemak)
    return encMessage_colemak
def dec(message, key, colekey):
    decMessage_colemak = cole_dec(message, colekey)
    print("-------- LEVEL 1 DECRYPTION : ---------")
    print(decMessage_colemak)
    print(" ")
    print("-------- LEVEL 2 DECRYPTION : ---------")
    decMessage = rsa.decrypt(decMessage_colemak, key).decode()
    print(decMessage)
    return decMessage
def cole_enc (mes, key) :
    mes = byte_to_hex(mes)
    enc_mes = colemak.encrypt(mes, key)
    return enc_mes

def cole_dec (mes, key) :
    enc_mes = colemak.decrypt(mes, key)
    enc_mes_bytes = hex_to_byte(enc_mes)
    return enc_mes_bytes
def byte_to_hex (byte) :
    hex_representation = binascii.hexlify(byte).decode()
    return hex_representation
```

```
def hex_to_byte (hex) :
    normal_byte_string = binascii.unhexlify(hex)
    return normal_byte_string
```

COLEMAK SUBSTITUTION CIPHER – INTERACTIVE GUI FOR CLIENT:

```
from tkinter import *
from socket import *
import _thread
from tkinter import simpledialog
import test_rsa_mod as trm
import rsa as rs
from datetime import datetime
import test as secret

def get_server_ip():
    return simpledialog.askstring("Server IP", "Enter Server IP Address:")

server_ip = str(get_server_ip())
print(str(server_ip))
# initialize server connection
def initialize_client():
    # initialize socket
    client_socket = socket(AF_INET, SOCK_STREAM)
    # config details of server
    host = server_ip
    port = 1234
    # connect to server
    client_socket.connect((host, port))
    return client_socket

# update the chat log
def update_chat(msg, state):
```

```
    global chatlog

    chatlog.config(state=NORMAL)
    # update the message in the window
    if state==0:
        chatlog.insert(END, datetime.now().strftime("%H:%M:%S") + ', YOU: ' + msg)
    else:
        chatlog.insert(END, datetime.now().strftime("%H:%M:%S") + ', SERVER: ' + msg)
    chatlog.config(state=DISABLED)
    # show the latest messages
    chatlog.yview(END)
serverkey, colekey = keyexchange()

# function to send message
def send():
    global textbox
    # get the message
    msg = textbox.get("0.0", END)
    # update the chatlog
    update_chat(msg, 0)
    newmsg = trm.enc(msg, serverkey, colekey)
    # send the message
    client_socket.send(newmsg.encode('ascii'))
    textbox.delete("0.0", END)

# function to receive message
def receive():
    while 1:
        try:
            data = client_socket.recv(1024)
            msg = data.decode('ascii')
            if msg != "":
                update_chat(trm.dec(msg, c_privatekey, colekey), 1)
```

```
        except:
            Pass
def press(event):
    send()

# GUI function
def GUI():
    global chatlog
    global textbox

    # initialize tkinter object
    gui = Tk()
    # set title for the window
    gui.title("Client Chat")
    # set size for the window
    gui.geometry("380x430")

    # Create a canvas for the background
    canvas = Canvas(gui, width=380, height=430)
    canvas.pack()

    # text space to display messages
    chatlog = Text(gui, bg='black', fg='white')
    chatlog.config(state=DISABLED)
# button to send messages
    sendbutton = Button(gui, bg='orange', fg='black', text='SEND', command=send)

    # textbox to type messages
    textbox = Text(gui, bg='grey', fg='black')

    chatlog.place(x=6, y=6, height=386, width=370)
    textbox.place(x=6, y=401, height=20, width=265)
    sendbutton.place(x=300, y=401, height=20, width=50)
```

```
    # bind textbox to use ENTER Key
    textbox.bind("<KeyRelease-Return>", press)

    # create thread to capture messages continuously
    _thread.start_new_thread(receive, ())

    # to keep the window in loop
    gui.mainloop()


if _name_ == '_main_':
    chatlog = textbox = None
    client_socket = initialize_client()
    GUI()
```

COLEMAK SUBSTITUTION CIPHER – INTERACTIVE GUI FOR SERVER:

```
from tkinter import *
from socket import *
import socket as sock
import _thread
from tkinter import simpledialog
import test_rsa_mod as trm
import rsa as rs
from datetime import datetime
import test as secret

def show_ip ():
    host = sock.gethostbyname(sock.gethostname())
    mess = f"Share this IP address with your client:\n\n{host}:\n\nPress OK when done !"
    simpledialog.messagebox.showinfo("Dialog Box", mess)

show_ip()
# initialize server connection
```

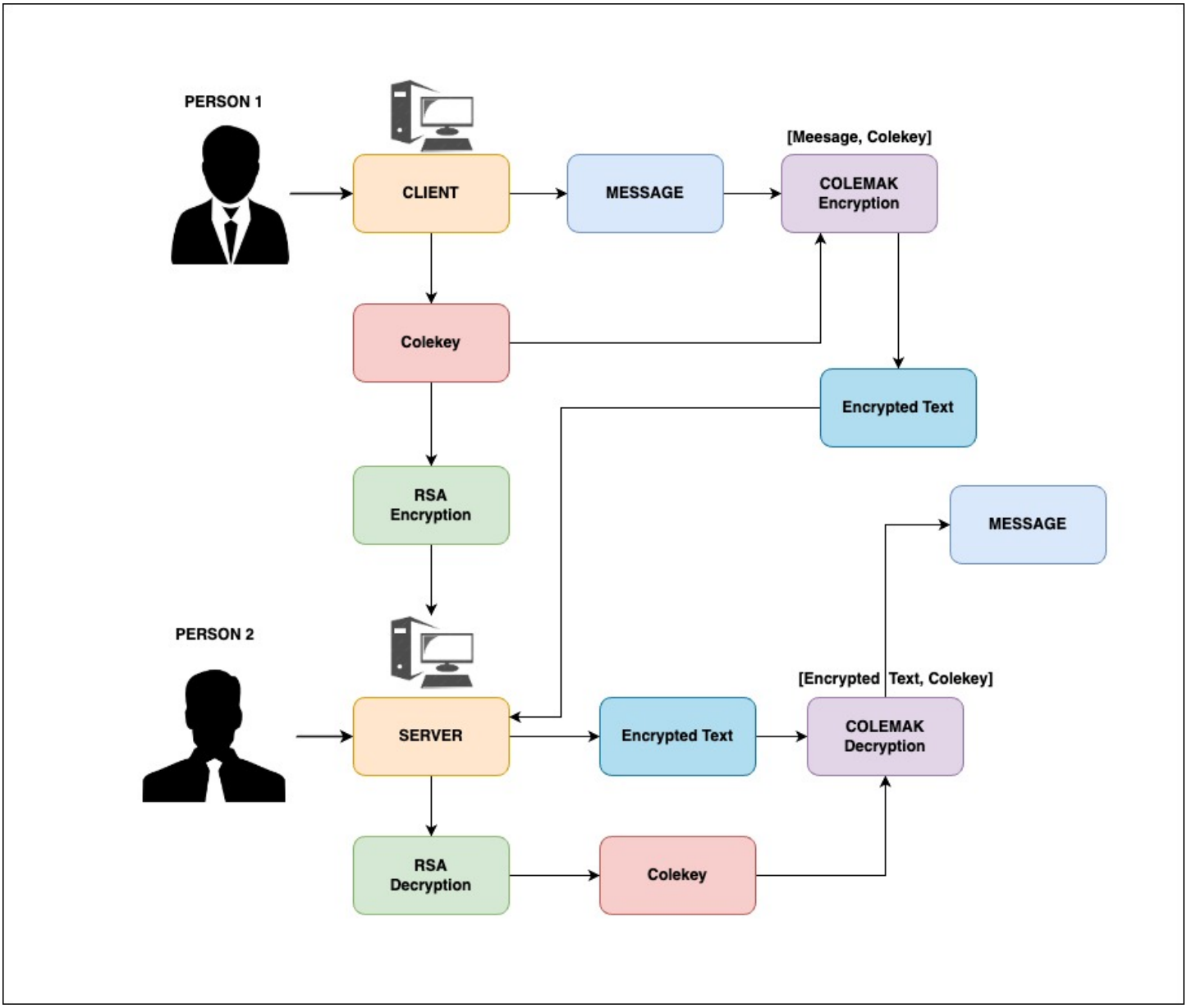
```
def initialize_server():
    # initialize socket
    s = socket(AF_INET, SOCK_STREAM)
    # config details of server
    host = sock.gethostbyname(sock.gethostname())
    port = 1234
    # initialize server
    s.bind((host, port))
    # set no. of clients
    s.listen(1)

    # accept the connection from client
    conn, addr = s.accept()
    return conn, addr
# update the chat log
def update_chat(msg, state):
    global chatlog
    chatlog.config(state=NORMAL)
    # update the message in the window
    if state==0:
        chatlog.insert(END, datetime.now().strftime("%H:%M:%S") + ', YOU: ' + msg)
    else:
        chatlog.insert(END, datetime.now().strftime("%H:%M:%S") + ', CLIENT: ' + msg)
    chatlog.config(state=DISABLED)
    # show the latest messages
    chatlog.yview(END)
# function to send message
def send():
    global textbox
    # get the message
    msg = textbox.get("0.0", END)
    # update the chatlog
```

```
    update_chat(msg, 0)
    newmsg = trm.enc(msg, clientkey, colekey)
    # send the message
    conn.send(newmsg.encode('ascii'))
    textbox.delete("0.0", END)

# function to receive message
def receive():
    while 1:
        data = conn.recv(1024)
        msg = data.decode('ascii')
        update_chat(trm.dec(msg, s_privatekey, colekey), 1)

def press(event):
    send()
# GUI function
def GUI():
    global chatlog
    global textbox

    # initialize tkinter object
    gui = Tk()
    # set title for the window
    gui.title("Server Chat")
    # set size for the window
    gui.geometry("380x430")

    # Create a canvas for the background
    canvas = Canvas(gui, width=380, height=430)
    canvas.pack()

    # text space to display messages
    chatlog = Text(gui, bg='black', fg='white')
```

```
    chatlog.config(state=DISABLED)

    # button to send messages
    sendbutton = Button(gui, bg='orange', fg='black', text='SEND', command=send)
# textbox to type messages
    textbox = Text(gui, bg='grey', fg='black')

    # place the components in the window
    chatlog.place(x=6, y=6, height=386, width=370)
    textbox.place(x=6, y=401, height=20, width=265)
    sendbutton.place(x=300, y=401, height=20, width=50)

    # bind textbox to use ENTER Key
    textbox.bind("<KeyRelease-Return>", press)

    # create thread to capture messages continuously
    _thread.start_new_thread(receive, ())

    # to keep the window in loop
    gui.mainloop()


if _name_ == '_main_':
    chatlog = textbox = None
    conn, addr = initialize_server()
    GUI()
```

*** The same has also been implemented in JAVA programming language.

*** This methodology supports data transfer only upto 53 bytes because of the keysize being 512. To make it support data regardless of its size, a simple change in the methodlogy will do. That methodology being supportive to any sizes of data, compromises in data privacy. This methodology is referred to as “methodology 2” hereby.



COLEMAK SUBSTITUTION CIPHER (METHODOLOGY 2) :

SERVER CODE :

```
import socket
import rsa as rs
import test as secret
import test_rsa_mod as rsa
import COLEMAK as colemak

def server_program():
    host = socket.gethostname() # CHANGE TO REQUIRED SERVER IP ADDRESS
    port = 5000

    s_publickey, s_privatekey = rsa.genkey()

    server_socket = socket.socket()
    server_socket.bind((host, port))
    server_socket.listen(2)
    conn, address = server_socket.accept()
    print("Connection from: " + str(address))

    num = 0
    cli_e = 0
    cli_n = 0

    while num <= 2 :
        if num == 0 :
            skey_n = s_publickey.n
            conn.send(str(skey_n).encode())
            cli_n = conn.recv(1024).decode()

        elif num == 1 :
```

```
            skey_e = s_publickey.e
            conn.send(str(skey_e).encode())
            print("Public key sent !")
            cli_e = conn.recv(1024).decode()
            print("Client's public key recieved")
            clientkey = rs.PublicKey(int(cli_n), int(cli_e))
            print("CLIENT PUBLIC KEY :", clientkey)
        elif num == 2 :
            enc_colekey = conn.recv(1024)
            colekey = rs.decrypt(enc_colekey, s_privatekey)
            colekey = str(colekey)[2:len(colekey)+2]
            print("Decrypted Colekey recieved")
        num = num+1

    while True:
        data = conn.recv(1024).decode()
        safe_data = colemak.decrypt(data, colekey)
        if not data:
            break
        print("from connected user (Encrypted) : ", data)
        print("from connected user (Decrypted) : ", safe_data)
        message = input(' -> ')
        safe_message = colemak.encrypt(message, colekey)
        conn.send(safe_message.encode())

    conn.close()

if _name_ == '_main_':
    server_program()
```

CLIENT CODE :

```
import socket
import rsa as rs
import test_rsa_mod as rsa
import os
import test as secret
import COLEMAK as colemak

def install_missing_libraries():
    required_libraries = ['psutil','hashlib','platform','os','socket','uuid','json','time','rsa']
    for library in required_libraries:
        try:
            _import_(library)
        except ImportError:
            print(f"{library} library not found. Installing...")
            os.system(f'pip install {library}')

install_missing_libraries()

def client_program():
    host = socket.gethostname() # CHANGE TO REQUIRED SERVER IP ADDRESS
    port = 5000
    c_publickey, c_privatekey = rsa.genkey()
    client_socket = socket.socket()
    client_socket.connect((host, port))
    num = 0
    ser_e = 0
    ser_n = 0

    while num <= 2 :
        if num == 0 :
            ckey_n = str(c_publickey.n)
```

```
            client_socket.send(str(ckey_n).encode())
            ser_n = client_socket.recv(1024).decode()

        elif num == 1 :
            ckey_e = str(c_publickey.e)
            client_socket.send(str(ckey_e).encode())
            print("Public key sent !")
            ser_e = client_socket.recv(1024).decode()
            print("Server's public key recieved")
            serverkey = rs.PublicKey(int(ser_n), int(ser_e))
            print("SERVER PUBLIC KEY :", serverkey)
            colekey = secret.generate_device_fingerprint()
            colekey = str(colekey)
        elif num == 2 :
            client_socket.send(rs.encrypt(colekey.encode(), serverkey))
            print("Encrypted Colekey sent")
        num = num+1

    message = input(" -> ")
    while message.lower().strip() != 'bye' and num != 0:
        safe_message = colemak.encrypt(message, colekey)
        client_socket.send(safe_message.encode())
        data = client_socket.recv(1024).decode()
        safe_data = colemak.decrypt(data, colekey)
        print('Received from server (Encrypted) : ', data)
        print('Received from server (Decrypted) : ', safe_data)
        message = input(" -> ")

    client_socket.close()

if _name_ == '_main_':
    client_program()
```

CONCLUSION:

In conclusion, the chat application embodies a robust fusion of asymmetric RSA and symmetric COLEMAK encryption techniques within a client-server architecture, ensuring a multi-layered approach to message security. While RSA encryption facilitates secure communication through the exchange of public and private keys, COLEMAK adds an additional layer of unpredictability, leveraging a unique keyboard layout-inspired substitution cipher bolstered by the device-specific colekey. This two-layer encryption process, where messages are successively encrypted and decrypted using the recipient's public and private keys, culminates in a highly secure and confidential communication environment. The methodology behind this application underscores the significance of the colekey as a unique device fingerprint, emphasizing the careful consideration given to security measures. With a user-friendly graphical interface, this application offers a seamless and intuitive platform for secure messaging, emphasizing the critical importance of combining multiple encryption layers for enhanced data protection and privacy. The second methodology while supporting data regardless of its size, compromises for the privacy of the same, due to which for a data transfer of small sizes, the first methodology is preferred.