PROJECT SYNOPSIS

INFORMATION SECURITY LAB

2025

EFFORTS BY:

AYUSH GUPTA (21103016)

SHREYA AGRAWAL (21103028)

KANISHK RAJ MITTAL (21103015)

PROJECT OVERVIEW:

In today's digital age, privacy and security have become paramount concerns in online communication. To address these concerns, our project aims to develop an end-to-end encryption and decryption of both audio and text data in chat applications. This system will ensure that only the intended recipients can access and understand the communicated information while preventing unauthorized parties from intercepting or deciphering the content.

OBJECTIVE:

Some of the objectives to be covered in our project are:-

1. Design and Develop Encryption Algorithms: Develop advanced encryption algorithms suitable for both audio and text data. This involves creating algorithms that provide strong encryption while maintaining reasonable performance to ensure real-time communication by minimizing any delays or lags in communication.
2. Implement Decryption Algorithms: Design and implement corresponding decryption algorithms to securely recover the original audio and text data on the recipient's side. These decryption algorithms should work seamlessly with the encryption algorithms while maintaining data integrity.
3. Key Management: Establish a robust key management system to generate, distribute, and securely store encryption keys. This includes mechanisms for key exchange between users to initiate secure communication sessions.

ALGORITHMS USED:

1) AES (Advanced Encryption Standard)

2) RSA (Rivest-Shamir-Adleman)

Incorporating these algorithms into our end-to-end chat encryption and decryption project will help us achieve robust security and will help us encrypt both text and audio files.

AES is a symmetric-key encryption algorithm known for its efficiency and strength, while RSA is an asymmetric-key algorithm often used for secure key exchange and digital signatures.

1) AES ENCRYPTION/DECRYPTION

AES can be used to encrypt both text and audio data. When a user sends a message or audio clip, your application would use a symmetric AES encryption key to encrypt the data. AES provides different key sizes (128-bit, 192-bit, or 256-bit) for varying levels of security.

2) RSA ENCRYPTION/DECRYPTION

Generating RSA Key Pairs:
Implement RSA key pair generation for each user. A public-private key pair is generated for each user. The public key is shared openly, while the private key is kept secret.

3) HYBRID ENCRYPTION

.

You can implement hybrid encryption, which combines the benefits of both symmetric and asymmetric encryption. Here's how the process might work:

This hybrid approach provides the efficiency of AES for bulk data encryption and the security benefits of RSA for secure key exchange.

PROGRAM CODE:

```
import socket
import threading
import pickle
from two_way_server import filepush
from two_way_server import receive_image
from functions import *
import time




stop_loops=1
reciver_port=5002 # server port



def send_messages(client_socket,mac_id):
    while True:
        message = input('You -> ')
        if(message[0:2]=="p "):
            d_type="photo"
            filename=message[2:]
            data=[filename,d_type,mac_id]
```

```
    serial_data=encrypt_serialized_data(data,client_public_key)
    client_socket.send(serial_data)
    send_image(client_socket,filename)
    filepush('Chat.txt','You-> '+filename+" Sent!");
    continue
elif(message==""):
    show_chat()
else:
    d_type="Text"
    send_data=message

    data=[send_data,d_type,mac_id]
    serial_data=encrypt_serialized_data(data,client_public_key)
    client_socket.send(serial_data)
    filepush('Chat.txt','You-> '+message)
show_chat()



if(message=="EXIT 0000"):
    stop_loops=0
    return
```

```
def sender_program(host,port,mac_id):
    input("Server Running->")
    sender_socket = socket.socket()
    sender_socket.connect((host, port))
    sender_socket.send(serialize_key(public_key))

    send_thread = threading.Thread(target=send_messages,
args=(sender_socket,mac_id))
    send_thread.start()
```

```
def receive_messages(client_socket,address):
    while stop_loops:
        deserial_data = client_socket.recv(1024)

        if not deserial_data:
            break
        name=str(address[0])
        data=decrypt_and_deserialize(deserial_data,private_key)
```

```
        data=list(data)


        rec_data=data[0]
        d_type=data[1]
        mac_=data[2]


        if(d_type=="photo"):


            filename=data[0]
            receive_image(client_socket,filename)
            filepush('Chat.txt',mac_+' :'+filename+" Recieved");


        else:
            text=rec_data
            filepush('Chat.txt',mac_+' :'+text);


    client_socket.close()


def reciver_program():# port in use 5001
    global client_public_key
    host = "0.0.0.0"
    port = reciver_port


    server_socket = socket.socket()
    server_socket.bind((host, port))
```

```
    server_socket.listen(5)


    conn, address = server_socket.accept()
    print('Connection from: ' + str(address))


    client_public_key=conn.recv(1024)


    client_public_key=deserialize_key(client_public_key)


    receive_thread = threading.Thread(target=receive_messages,
args=(conn,address))
    receive_thread.start()









if __name__=='__main__':
    mac_id=get_mac_address()
    my_ip_address=get_local_ip()
    print("IP ADDRESS:",my_ip_address)
    port=5001
    # host = "192.168.184.176"
    host=input("Enter Host IP:")
```

```
    private_key, public_key  =generate_key_pair()
    client_public_key=0
    sender=threading.Thread(target=sender_program,args=(host,port,mac_id))
    reciver=threading.Thread(target=reciver_program,args=())
    sender.start()
    reciver.start()
```

Functions.py

```
import pickle
import uuid
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.primitives.asymmetric import rsa, padding
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.backends import default_backend
import os, platform
import socket
from tqdm import tqdm




def send_image(client_socket, filename):
    file_size = os.path.getsize(filename)
```

```
    with open(filename, 'rb') as file:
        with tqdm(total=file_size, unit='B', unit_scale=True, desc="Sending",
ncols=80) as pbar:
            while True:
                data = file.read(1024)
                if not data:
                    break
                client_socket.send(data)
                pbar.update(len(data))
    # Close the file after sending
    file.close()




def receive_image(server_socket, filename):
    with open(filename,'wb') as file:
        while True:
            data = server_socket.recv(1024)
            if not data:
                break
            file.write(data)
        file.close()
```

```
def serialize(data):
    try:
        serialized_data = pickle.dumps(data)
        return serialized_data
    except Exception as e:
        print(f"Error during serialization: {str(e)}")
        return None

def deserialize(serialized_data):
    try:
        data = pickle.loads(serialized_data)
        return data
    except Exception as e:
        print(f"Error during deserialization: {str(e)}")
        return None

def get_mac_address():
    mac = uuid.UUID(int=uuid.getnode()).hex[-12:]
    return ':'.join([mac[e:e+2] for e in range(0, 12, 2)])



def generate_key_pair():
    private_key = rsa.generate_private_key(
        public_exponent=65537,
        key_size=2048,
```

```
        backend=default_backend()
    )
    public_key = private_key.public_key()
    return private_key, public_key




def serialize_key(key):
    try:
        # If the key is a private key, extract the public key
        if isinstance(key, rsa.RSAPrivateKey):
            key = key.public_key()

        # Serialize the public key to PEM format
        serialized_key = key.public_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PublicFormat.SubjectPublicKeyInfo
        )
        return serialized_key
    except Exception as e:
        print(f"Error in serialize_key: {e}")
        return None

def deserialize_key(server_public_key_data):
    server_public_key =
```

```
serialization.load_pem_public_key(server_public_key_data,
backend=default_backend())
    return server_public_key




def encrypt_message(message, public_key):
    cipher_text = public_key.encrypt(
        message.encode(),
        padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None
        )
    )
    return cipher_text

def decrypt_message(cipher_text, private_key):
    try:
        decrypted_message = private_key.decrypt(
            cipher_text,
            padding.OAEP(
                mgf=padding.MGF1(algorithm=hashes.SHA256()),
                algorithm=hashes.SHA256(),
                label=None
            )
```

```
        )
        return decrypted_message.decode()
    except Exception as e:
        print(f"Error in decrypt_message: {e}")
        return None

def clear_terminal():
    system_platform = platform.system().lower()
    if system_platform == "windows":
        os.system("cls")
    else:
        os.system("clear")

def show_chat():
    clear_terminal()
    file_path="Chat.txt"
    try:
        with open(file_path, 'r') as file:
            content = file.read()
            print(content)
    except FileNotFoundError:
        print(f"Error: File '{file_path}' not found.")
    except Exception as e:
        print(f"Error: {e}")
```

```
def encrypt_serialized_data(data, public_key):
    try:
        # Serialize the data
        serialized_data = pickle.dumps(data)

        # Encrypt the serialized data
        encrypted_data = public_key.encrypt(
            serialized_data,
            padding.OAEP(
                mgf=padding.MGF1(algorithm=hashes.SHA256()),
                algorithm=hashes.SHA256(),
                label=None
            )
        )

        return encrypted_data

    except Exception as e:
        print(f"Error in encrypt_serialized_data: {e}")
        return None

def decrypt_and_deserialize(encrypted_data, private_key):
    try:
        # Decrypt the data
        decrypted_data = private_key.decrypt(
```

```
            encrypted_data,
            padding.OAEP(
                mgf=padding.MGF1(algorithm=hashes.SHA256()),
                algorithm=hashes.SHA256(),
                label=None
            )
        )

        # Deserialize the decrypted data
        original_data = pickle.loads(decrypted_data)

        return original_data

    except Exception as e:
        print(f"Error in decrypt_and_deserialize: {e}")
        return None

def save_text_to_file(file_path, text):
    try:
        with open(file_path, 'w', encoding='utf-8') as file:
            file.write(text)
        print(f"Text saved successfully to {file_path}")
    except Exception as e:
        print(f"Error saving text to {file_path}: {e}")
```

```
def get_local_ip():
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        s.connect(("8.8.8.8", 80))
        local_ip = s.getsockname()[0]
        return local_ip
    except Exception as e:
        print(f"Error getting local IP: {e}")
        return None


def send_file(client_socket, file_path):
    file_size = os.path.getsize(file_path)
    client_socket.send(f"{file_size}".encode("utf-8"))  # Send the file size


    with open(file_path, 'rb') as file:
        with tqdm(total=file_size, unit='B', unit_scale=True, desc="Sending",
ncols=80) as pbar:
            while True:
                data = file.read(1024)
                if not data:
                    break
                client_socket.send(data)
                pbar.update(len(data))


    print("File sent successfully!")
```

```
def receive_file(server_socket, save_path):
    file_size = int(server_socket.recv(1024).decode("utf-8"))
    received_size = 0

    with open(save_path, 'wb') as file:
        with tqdm(total=file_size, unit='B', unit_scale=True, desc="Receiving",
ncols=80) as pbar:
            while received_size < file_size:
                data = server_socket.recv(1024)
                file.write(data)
                received_size += len(data)
                pbar.update(len(data))

    print("File received successfully!")
```