Network Security (CSE 350)

Assignment-3

Prof. B.N. Jain

Project 1 : RSA-based Public key Distribution Authority (PKDA)
Algorithm:
(1363+1362)%2=1

Submitted by:

Name	Roll No
Vansh	2021363
V. Bharath Krishna	2021362

Brief Overview of Project

Programming Language: C++
Platform: MS Windows

In this project we have builded a PKDA, that responds to clients that seek their own public-keys from the PKDA authority system which have all of the public keys considering for client A and client B both, when either of the client requests for a public key, the PKDA will encrypts the requested public key and sends it back to the client, where as in the client system they will generate their own RSA- keys (both public and private keys) and uses the private key for decrypting the messages that has been sent from either Client or PKDA

In which we have used socket programming for the PKDA - client , client -client communication by running up three terminals for both of the clients and PKDA for setting up the connection and following up the communication

RSA implementation

We have implemented an RSA system with its functionalities like encrypting, decrypting, Generating Keys, Computing the decryption key. The RSA funcitons are incorported in the client and PKDA codes. We also have functions like gcd which calculates the greatest common divisor which can be used to check the decryption key. ModInverse Function is used to calculate the value of decryption key. Generate Keys Function which takes p and q as input and returns a pair of key, one for encrypting and the other for decrypting. Encrypt and Decrypt function Encrypt and decrypt a character sent by encrypt_algo and decrypt_algo function respectively.

Encryption Function

For encryption using a key (e, n) we follow the following rule:

C=pow(M,e)%n

An optimized approach is used to calculate C instead of calculating a large number pow(M,e).
In each step while multiplying M to some value, we take its modulo with n. Each

Decryption Function

For Decryption using a key (d, n) we follow the following rule:

M=pow(C,d)%n

An optimized approach is used to calculate C instead of calculating a large number pow(M,d).
In each step while multiplying M to some value, we take its modulo with n.

PKDA, Client A and Client B:

We have implemented PKDA, Client A, and Client B using socket programming, where each of them act as client and server, waiting for a connection and also sending messages to others. We have accomplished this task by using threads. It is assumed that PKDA has public keys of all clients. Clients have their public, private key and public key of the PKDA. In our system, clients at time of registering in PKDA, generate their keys and register on PKDA with their public key. A client can seek for any other client's or its own public key from PKDA. There are total 3 code files, one for each. We uniquely identify clients by their port no. Which in our case is also used as their ID. We calculate nonce using Rand() function and current time and duration using chrono library.

PKDA Implementation

encrypt(plaintext, e, n): This function performs RSA encryption on a plaintext using the provided public key (e, n). It repeatedly multiplies the plaintext by itself e times modulo n to produce the ciphertext.

decrypt(ciphertext, d, n): This function performs RSA decryption on a ciphertext using the provided private key (d, n). It repeatedly multiplies the ciphertext by itself d times modulo n to retrieve the original plaintext.

handle_connection(client_socket): This function handles the connection with a client. It runs in an infinite loop, receiving data from the client. When it receives a message, it checks its type. If the message is a "REQUEST_PUBLIC_KEY" message, it extracts the client ID from the message, looks up the corresponding public key in the clientPublicKeys dictionary, and sends it to the client. If the client ID is not found, it sends a "CLIENT_NOT_FOUND" message back to the client.

start_server(port): This function initializes the server socket, binds it to the specified port on all available network interfaces, and starts listening for incoming connections. When a client connects, it spawns a new thread to handle the connection using the handle_connection function.

Client Implementation

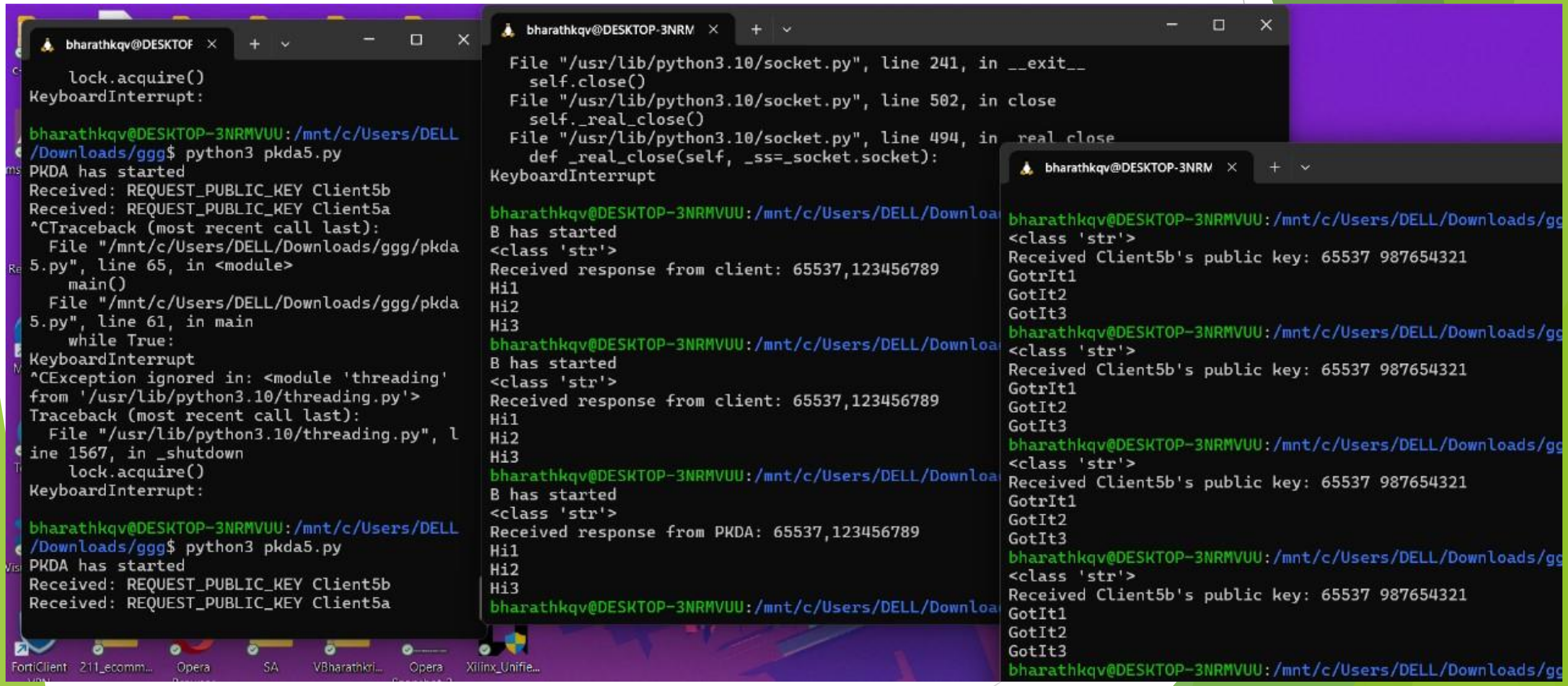
encrypt(plaintext, e, n): This function encrypts a plaintext message using RSA encryption. It takes the plaintext message, public exponent e, and modulus n as parameters. It iterates e times, multiplying the plaintext with itself modulo n in each iteration to compute the ciphertext.

decrypt(ciphertext, d, n): This function decrypts a ciphertext message using RSA decryption. It takes the ciphertext message, private exponent d, and modulus n as parameters. It iterates d times, multiplying the ciphertext with itself modulo n in each iteration to retrieve the original plaintext.

send_message(message, ip, port): This function sends a message to a specified IP address and port using a TCP socket connection. It takes the message, destination IP address, and port number as parameters. It establishes a socket connection, sends the message encoded as bytes, and waits for a response. If a response is received, it prints the response.

main(): This function serves as the entry point of the script. It initializes variables such as the IP address and port of the PKDA server and the private key exponent (d) of a client (Client5a). Then, it establishes a TCP connection with the PKDA server, sends a request for the public key of Client5b, and receives and parses the public key response. Afterward, it enters a message exchange loop where it encrypts a message using Client5b's public key and sends it to Client5b.

Output


bharathkqv@DESKTOP
lock.acquire()
KeyboardInterrupt:
bharathkqv@DESKTOP-3NRMVUU:/mnt/c/Users/DELL/Downloads/ggg$ python3 pkda5.py
PKDA has started
Received: REQUEST_PUBLIC_KEY Client5b
Received: REQUEST_PUBLIC_KEY Client5a
^CTraceback (most recent call last):
File "/mnt/c/Users/DELL/Downloads/ggg/pkda5.py", line 65, in <module>
main()
File "/mnt/c/Users/DELL/Downloads/ggg/pkda5.py", line 61, in main
while True:
KeyboardInterrupt
^CException ignored in: <module 'threading' from '/usr/lib/python3.10/threading.py'>
Traceback (most recent call last):
File "/usr/lib/python3.10/threading.py", line 1567, in _shutdown
lock.acquire()
KeyboardInterrupt:
bharathkqv@DESKTOP-3NRMVUU:/mnt/c/Users/DELL/Downloads/ggg$ python3 pkda5.py
PKDA has started
Received: REQUEST_PUBLIC_KEY Client5b
Received: REQUEST_PUBLIC_KEY Client5a
bharathkqv@DESKTOP-3NRM
File "/usr/lib/python3.10/socket.py", line 241, in __exit__
self.close()
File "/usr/lib/python3.10/socket.py", line 502, in close
self._real_close()
File "/usr/lib/python3.10/socket.py", line 494, in _real_close
def _real_close(self, _ss=_socket.socket):
KeyboardInterrupt
bharathkqv@DESKTOP-3NRMVUU:/mnt/c/Users/DELL/Downloa
B has started
<class 'str'>
Received response from client: 65537,123456789
Hi1
Hi2
Hi3
bharathkqv@DESKTOP-3NRMVUU:/mnt/c/Users/DELL/Downloa
B has started
<class 'str'>
Received response from client: 65537,123456789
Hi1
Hi2
Hi3
bharathkqv@DESKTOP-3NRMVUU:/mnt/c/Users/DELL/Downloa
B has started
<class 'str'>
Received response from PKDA: 65537,123456789
Hi1
Hi2
Hi3
bharathkqv@DESKTOP-3NRMVUU:/mnt/c/Users/DELL/Downloa
bharathkqv@DESKTOP-3NRM
bharathkqv@DESKTOP-3NRMVUU:/mnt/c/Users/DELL/Downloads/gg
<class 'str'>
Received Client5b's public key: 65537 987654321
GotrIt1
GotIt2
GotIt3
bharathkqv@DESKTOP-3NRMVUU:/mnt/c/Users/DELL/Downloads/gg
<class 'str'>
Received Client5b's public key: 65537 987654321
GotrIt1
GotIt2
GotIt3
bharathkqv@DESKTOP-3NRMVUU:/mnt/c/Users/DELL/Downloads/gg
<class 'str'>
Received Client5b's public key: 65537 987654321
GotrIt1
GotIt2
GotIt3
bharathkqv@DESKTOP-3NRMVUU:/mnt/c/Users/DELL/Downloads/gg
<class 'str'>
Received Client5b's public key: 65537 987654321
GotIt1
GotIt2
GotIt3
bharathkqv@DESKTOP-3NRMVUU:/mnt/c/Users/DELL/Downloads/gg
FortiClient
211_ecomm...
Opera
SA
VBharathkri...
Opera
Xilinx_Unifie...