Project no. 1: RSA-based Public Key Distribution Authority (PKDA)

Programming language used: Python
Platform: MS Windows/ Virtual Machine

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

RSA Implementation:

- **gcd(int a, int b):** This function calculates the greatest common divisor (GCD) of two integers a and b using Euclid's algorithm.
- **modInverse(int a, int m):** This function calculates the modular multiplicative inverse of a modulo m using brute force by checking each possible value until finding the one that satisfies the condition (a * x) % m == 1.
- **generateKeys(int p, int q, int& publicKey, int& privateKey):** This function generates RSA public and private keys based on two prime numbers p and q. It calculates n as the product of p and q, and phi as (p - 1) * (q - 1). Then, it selects a public key e that is coprime with phi, and finds the corresponding private key d such that (e * d) % phi == 1.
- **encrypt(int plaintext, int e, int n):** This function encrypts a plaintext integer using the RSA algorithm. It repeatedly multiplies the plaintext by itself e times modulo n.
- **decrypt(int ciphertext, int d, int n):** This function decrypts a ciphertext integer using the RSA algorithm. It repeatedly multiplies the ciphertext by itself d times modulo n.
- **encrypt_algo(string plaintext, int e, int n):** This function encrypts a plaintext string using RSA encryption. It converts each character of the plaintext to its ASCII value, encrypts it using the encrypt function, and stores the ciphertext integers in a vector.
- **decrypt_algo(vector ciphertext, int d, int n):** This function decrypts a vector of ciphertext integers using RSA decryption. It decrypts each integer using the decrypt function and converts them back to characters to form the plaintext string.
- **main():** This is the main function of the program. It generates RSA keys using the generateKeys function with prime numbers 61 and 53, then prints the public key e and private key d to the console.

PKDA Implementation:

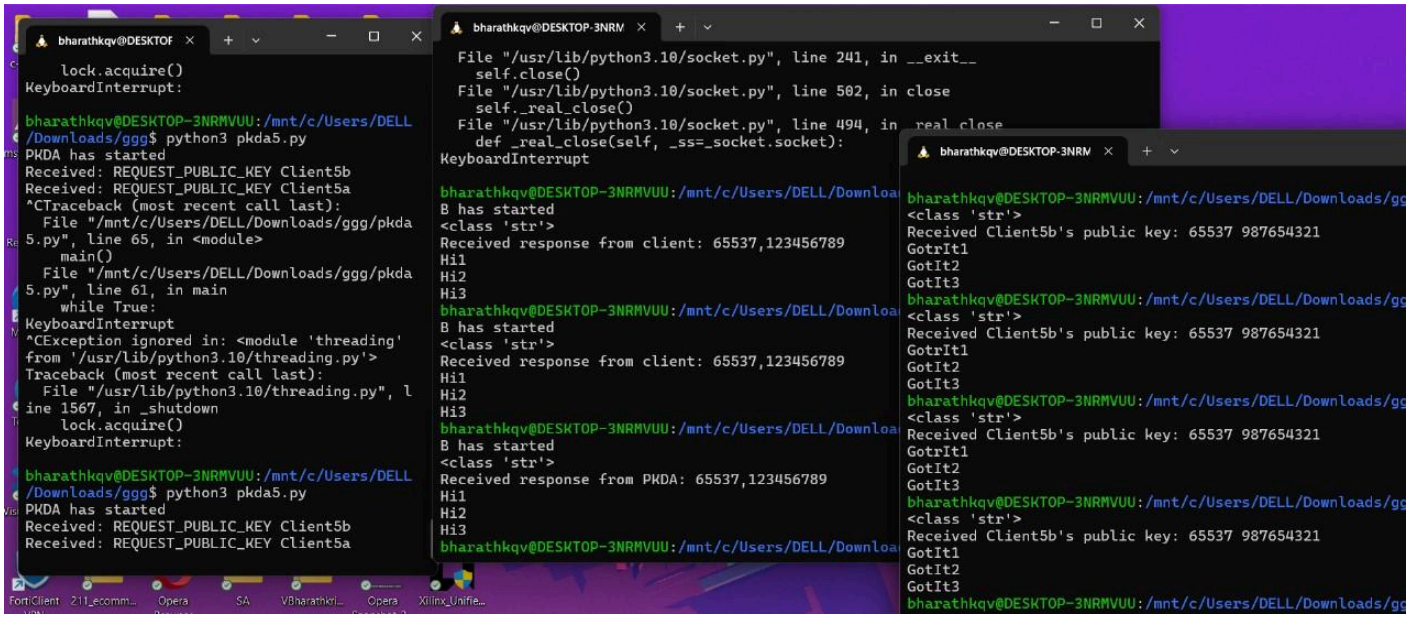
- **encrypt(plaintext, e, n):** This function performs RSA encryption on a plaintext using the provided public key (e, n). It repeatedly multiplies the plaintext by itself e times modulo n to produce the ciphertext.

- **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.
- **main():** This function serves as the entry point of the program. It prints a message indicating that the PKDA has started, then starts the server on port 5000 in a separate thread. It runs in an infinite loop, keeping the PKDA server running indefinitely.
- **if __name__ == "__main__":** This conditional block ensures that main() is executed only when the script is run directly, not when it's imported as a module into another script. It prevents the server from starting unintentionally when imported.

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.

The following inputs and 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...