Project 1

Siddhant Rana 2021101

Mahansh Aditya 2021334

STEPS TO RUN-

1)-First run pkda.py then run both files rana .py and mahansh.py (FIRST RUN MAHANSH .PY and enter the required field BECAUSE IT INITIATES THE SOCKET SERVER THEN RUN THE RANA.PY)

2)-Enter which one is the initiator and responder (ASSUMING ONE HAS TO BE INITIATOR AND THE OTHER RESPONDER)

3)-THE KEY EXCHANGE WILL BE STARTED AUTOMATICALLY BY THE INITIATOR .

4)-ENTER THE MESSAGE YOU WANT TO SEND TO THE OTHER CLIENT.

Overview

We successfully built a public key distribution system from scratch. The system consists of a Public Key Distribution Authority (PKDA) and multiple clients. PKDA serves as the central entity that manages and distributes public keys for all clients. Clients can request public keys from PKDA, either for themselves or for other clients.

Each client has its own key pair, consisting of a private key and a public key. Clients securely exchange messages with each other by encrypting messages using the recipient's public key and their own private key.

For encryption and decryption we are using RSA, we have also built it from scratch. The server (PKDA) sends back a response to the client by encrypting the data using the server's private key, which the client can decrypt using the server's public key.

For every message between client & server and client and client, we are sending nonce and timestamp and duration for which the message is valid to prevent replay attack. On receiving every response we are checking if the message arrived within the allowed timeframe and if the nonce received is unique or not.

And for server & client and client & client interaction we are using sockets.

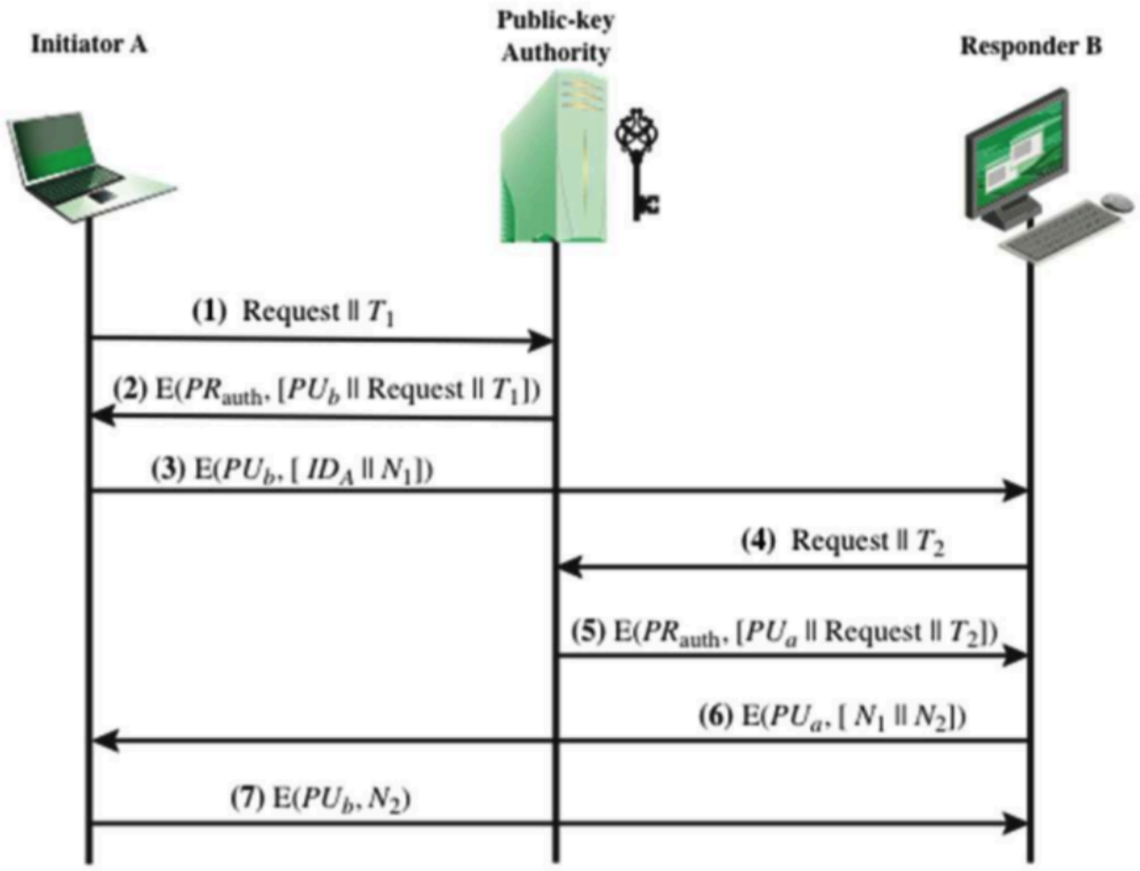
In a test scenario, Client A requested the public key of Client B from PKDA. After obtaining each other's public keys, Client A successfully sent encrypted messages ("Hi1", "Hi2", "Hi3") to Client B, who then decrypted and responded with encrypted messages ("Got-it1", "Got-it2", "Got-it3") to Client A.

Assumptions made:

1. PKDA has the public key of all the clients, clients don't need to register with PKDA.
2. Client already knows the public key of PKDA, they don't need to ask the server for its public key.
3. Client already knows their own public and private key, but not of the other client.
4. The values of nonce 1 and nonce2 might get interchanged because while coming back nonce1 will become nonce 2 in out code

Flow of the Code

We are following the flow mentioned in the pdf:


Initiator A
Public-key Authority
Responder B
(1) Request ‖ T_1
(2) E(PR_{auth}, [PU_b ‖ Request ‖ T_1])
(3) E(PU_b, [ID_A ‖ N_1])
(4) Request ‖ T_2
(5) E(PR_{auth}, [PU_a ‖ Request ‖ T_2])
(6) E(PU_a, [N_1 ‖ N_2])
(7) E(PU_b, N_2)

1. First client A sends a request to the server requesting the public key of client B. In this response, we send the identifier of A and and nonce and timestamp and nonce.
2. On receiving the client 's request, the server first checks if it's a valid request or not, by checking if the message arrived in the allowed timeframe and if the nonce is unique or not.
3. After validating that the response is unique, server then fetches the client B's public key, constructs the message by concatenating the public key of B, request which client sent and , timestamp which client send and encrypt this data using RSA algorithm with server's private key.
4. On receiving the response from server, client A decrypts the message using server's public key.

5. Then client A gets ready to send a message to client B. The message to client B consists of an identifier of A and nonce1. After constructing the message client A encrypts this data using client B’s public key.
6. At this point Client B , first receives client A’s message. After this , the client decrypts the message using its own private key and prints what it received, that is an identifier of A and nonce1.
7. Then client B requests the public key of client A to the server. Client B constructs the message which consists of timestamp , duration it’s valid , nonce and identifier of B.
8. On receiving the client 's request, the server first checks if it’s a valid request or not, by checking if the message arrived in the allowed timeframe and if the nonce is unique or not.
9. After validating that the response is unique, server then fetches the client A’s public key, constructs the message by concatenating the public key of A, request which client sent and , timestamp which client send and encrypt this data using RSA algorithm with server’s private key.
10. On receiving the response from server, client B decrypts the message using server’s public key.
11. Then client B gets ready to send a message to client A. The message to client B consists of nonce1 (which is the nonce that client A sent initially) and nonce2. After constructing the message client B encrypts this data using client ’s public key.
12. On receiving client B’s message, Client A encrypts the message , which consists of the nonce2 , which client received form the message sent by B, and then encrypts using client B’s public key and sends it to B.
13. On receiving client A’s message, client B decrypts it and prints it, which is just the nonce sent by client B earlier.
14. After this client A sends a message to B,saying ‘Hi1” along with the timestamp, duration and nonce , by encrypting it using client B’s public key.
15. Client B receives it, decrypts it and prints it. For response, client B sends back “Got 1’ along with the timestamp, duration and nonce , by encrypting it using client A’s public key.
16. This goes on 3 times.

Explanation of each function

generate_nonce :

1. Takes a string entity ('mine' or 'others') to specify for whom the nonce is being generated.

2. Generates a unique random nonce between 1 and 10000, ensuring it hasn't been used before for the specified entity, and returns it.

server.construct_message :

1. Takes a string sender representing the client's name.

2. Constructs a message data dictionary containing sender's name, current timestamp, duration (set to 300 seconds), and a unique nonce generated for the sender and returns it.

validate_response:

1. Takes a dictionary response received from the server.
2. Validates the server's response by checking for errors, nonce uniqueness, and timestamp within the valid duration (300 seconds).
3. Returns a boolean indicating the validity of the response (True if valid, False otherwise).

receive_key:

1. Constructs a message to request the public key , sends the request to the server, and receives the encrypted public key in response, decrypting it using PKDA's public key.
2. Updates the keyList dictionary with the received public key for the specified client.

receive_message:

1. Takes an integer step indicating the current step or phase of message exchange.
2. Receives an encrypted message from the client socket, decrypts it using the client's private key, and returns the decrypted message as a dictionary.

3. Returns the decrypted message data dictionary.

send_message:

1. Takes an integer step indicating the current step, an optional nonce, and an optional message to be sent.
2. Constructs a message containing sender's details, nonce, and optionally a nonce or message, encrypts it using the recipient's public key, and sends it through the client socket.
3. Sends the encrypted message to the recipient.

client_handler:

1. Takes client_socket and client_address representing the socket connection and address of the connected client.
2. Handles the client's connection by receiving a request, validating it, retrieving the requested public key, encrypting the response, and sending it back to the client.
3. Sends an encrypted response to the client or closes the connection in case of errors.

server.run(self):

1. Runs the server in a continuous loop, accepting incoming client connections, and creating a new thread for each client to handle client requests concurrently.
2. Continuously listens for incoming client connections and handles them using the client_handler method.

server.construct_message:

1. Takes status (string indicating the message status), public_key (public key associated with the client), and old_request (previous message data dictionary).
2. Constructs a message data dictionary containing status, public key, old message details, timestamp, duration (set to 300 seconds), and a unique nonce.
3. Returns the constructed message data dictionary.

get_public_key:

1. Takes client_id (identifier for the client), and old_request (previous message data dictionary).
2. Retrieves the public key associated with the given client_id, constructs a success message containing the public key and old request details, and returns the constructed message.
3. Returns a message data dictionary containing the public key and old request details as a response.

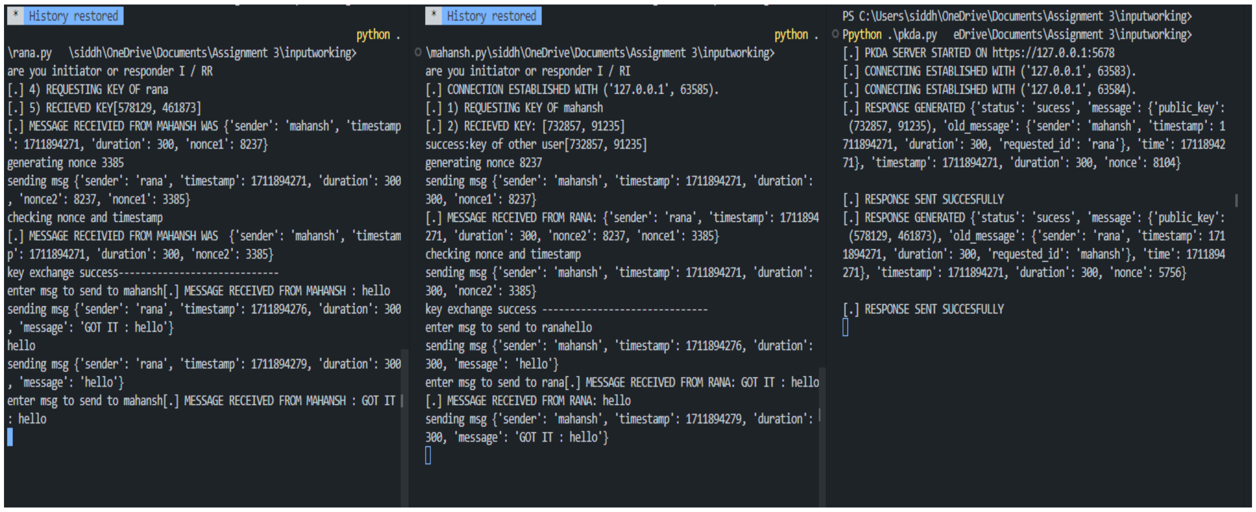
clientA.run:

1. Manages the client's communication flow by accepting a connection, receiving a public key, sending and receiving messages with specific steps, and validating the received messages.
2. Facilitates message exchange between clients and prints received messages and validation results.

clientB.run:

1. Orchestrates the client's communication sequence by receiving an initial message, obtaining a public key, sending and receiving messages with specific step identifiers, and validating the received messages.
2. Coordinates the message exchange between clients and prints received messages along with validation results.

Output



```
* History restored
                                                              python .
\rana.py   \siddh\OneDrive\Documents\Assignment 3\inputworking>
are you initiator or responder I / RR
[.] 4) REQUESTING KEY OF rana
[.] 5) RECIEVED KEY[578129, 461873]
[.] MESSAGE RECEIVIED FROM MAHANSH WAS {'sender': 'mahansh', 'timestamp
': 1711894271, 'duration': 300, 'nonce1': 8237}
generating nonce 3385
sending msg {'sender': 'rana', 'timestamp': 1711894271, 'duration': 300
, 'nonce2': 8237, 'nonce1': 3385}
checking nonce and timestamp
[.] MESSAGE RECEIVIED FROM MAHANSH WAS  {'sender': 'mahansh', 'timestam
p': 1711894271, 'duration': 300, 'nonce2': 3385}
key exchange success----------------------------
enter msg to send to mahansh[.] MESSAGE RECEIVED FROM MAHANSH : hello
sending msg {'sender': 'rana', 'timestamp': 1711894276, 'duration': 300
, 'message': 'GOT IT : hello'}
hello
sending msg {'sender': 'rana', 'timestamp': 1711894279, 'duration': 300
, 'message': 'hello'}
enter msg to send to mahansh[.] MESSAGE RECEIVED FROM MAHANSH : GOT IT
: hello
```

```
* History restored
                                                              python .
\mahansh.py\siddh\OneDrive\Documents\Assignment 3\inputworking>
are you initiator or responder I / RI
[.] CONNECTION ESTABLISHED WITH ('127.0.0.1', 63585).
[.] 1) REQUESTING KEY OF mahansh
[.] 2) RECIEVED KEY: [732857, 91235]
success:key of other user[732857, 91235]
generating nonce 8237
sending msg {'sender': 'mahansh', 'timestamp': 1711894271, 'duration':
300, 'nonce1': 8237}
[.] MESSAGE RECEIVED FROM RANA: {'sender': 'rana', 'timestamp': 1711894
271, 'duration': 300, 'nonce2': 8237, 'nonce1': 3385}
checking nonce and timestamp
sending msg {'sender': 'mahansh', 'timestamp': 1711894271, 'duration':
300, 'nonce2': 3385}
key exchange success ----------------------------
enter msg to send to ranahello
sending msg {'sender': 'mahansh', 'timestamp': 1711894276, 'duration':
300, 'message': 'hello'}
enter msg to send to rana[.] MESSAGE RECEIVED FROM RANA: GOT IT : hello
[.] MESSAGE RECEIVED FROM RANA: hello
sending msg {'sender': 'mahansh', 'timestamp': 1711894279, 'duration':
300, 'message': 'GOT IT : hello'}
```

```
PS C:\Users\siddh\OneDrive\Documents\Assignment 3\inputworking>
Ppython .\pkda.py   eDrive\Documents\Assignment 3\inputworking>
[.] PKDA SERVER STARTED ON https://127.0.0.1:5678
[.] CONNECTING ESTABLISHED WITH ('127.0.0.1', 63583).
[.] CONNECTING ESTABLISHED WITH ('127.0.0.1', 63584).
[.] RESPONSE GENERATED {'status': 'sucess', 'message': {'public_key':
 (732857, 91235), 'old_message': {'sender': 'mahansh', 'timestamp': 1
711894271, 'duration': 300, 'requested_id': 'rana'}, 'time': 17118942
71}, 'timestamp': 1711894271, 'duration': 300, 'nonce': 8104}

[.] RESPONSE SENT SUCCESFULLY
[.] RESPONSE GENERATED {'status': 'sucess', 'message': {'public_key':
 (578129, 461873), 'old_message': {'sender': 'rana', 'timestamp': 171
1894271, 'duration': 300, 'requested_id': 'mahansh'}, 'time': 1711894
271}, 'timestamp': 1711894271, 'duration': 300, 'nonce': 5756}

[.] RESPONSE SENT SUCCESFULLY
```