Chenhao Tan:
Do the connection between client and host,negotiate and establish a secure channel using the SSL/SSH handshake protocol.

Detailed implementation analysis of the client_program

The client program is fundamentally designed to communicate securely with the server side and perform banking operations. My implementation involves several levels of details which are key aspects of implementing a secure communication protocol:

1. Initialization and connection establishment:
   The client creates a new socket object with socket.socket() and then uses the connect method to connect to the IP address and port of the server. Here use socket.gethostname() to get the server address, usually used in the test environment, the actual deployment should specify the specific server IP.
2. The implementation of encrypted communication:
   The initial "hello" message is RSA encrypted with client_encrypt_rsa, which means you have set up the public and private RSA keys on the client side. The purpose of the encryption is to secure the data while it is being transmitted over the public network from being intercepted and tampered with by unauthorized persons.
   The server response received by the client is decrypted via client_decrypt_rsa to obtain a portion of the server's public key, which is used for subsequent message encryption to ensure that

only the specific server can decrypt the data sent by the client.

3.Authentication and session confirmation:

Sending a "prove it" message asking the server for proof of identity is an important step in ensuring that the other party is the intended server, increasing the security of the communication.

Next, the client decrypts the server's response using previously obtained information about the server's public key. The process of verifying the server's identity is critical to secure communication.

4.Business Operation Processing:

The client program provides a simple text interface that gives the user the option of depositing, withdrawing, or checking the balance. Each operation requires the user to enter the appropriate opcode and, in the case of deposits and withdrawals, the amount.

After each operation, the client encrypts the user's request and sends it to the server and waits for an encrypted response from the server before decrypting it and displaying it to the user. This interactive operation process makes the user experience more friendly and at the same time ensures the security of the operation.

5.Session Maintenance and Closing:

After the user has completed all operations, the session can be ended by typing '4' and the client will close the socket connection, gracefully ending communication with the server. This is very important in network programming to prevent port hogging and resource leakage.

Detailed implementation analysis of server_program

The server program is at the heart of responding to client requests. My implementation shows how encrypted communication is handled and how the appropriate banking operations are performed based on client requests.

1.Port binding and listening:

The server binds the socket to the local host and the specified port (6001) via socket.bind, and then starts listening on this port via the listen method. This is a common setup step for web servers and allows the server to receive connection requests from any client.

2.Handling connections and requests:

Connection requests from clients are accepted via the accept method. This step is blocking, i.e. the server waits here until a client initiates a connection.

Receives and parses the encrypted data sent by the client, which the server needs to be able to decrypt correctly and understand in order to respond appropriately.

3.Perform specific operations:

Depending on the type of request from the client (deposit, withdrawal, balance inquiry), the server calls the appropriate processing functions. These operations modify the global variable balance and encrypt the results and send them back to the client.

Using global variables to keep track of balance status is a simple state management approach for this type of single-threaded server simulation.

4.Security and Data Integrity:

All requests from the client and responses from the server are encrypted and decrypted via RSA, ensuring the security and integrity of the data in transit. This is especially important for applications involving monetary operations.

5.Session Management:

The server closes the connection after processing all requests from the client. This includes the server closing the socket connection correctly after the client sends a

session termination command to ensure that all resources are released to avoid potential resource leaks or security issues.

```
def client_program():
    host = socket.gethostname()
    port = 6001
    client_socket = socket.socket()
    client_socket.connect((host, port))
    #链接
    message="hello"
    client_socket.send(client_encrypt_rsa(message))
    data=client_socket.recv(1024)
    temp=client_decrypt_rsa(data)
    split = temp.split()
    server_public_key_part_2 = split[-1]
    server_public_key_part_1 = split[-2]
    message="prove it"
    client_socket.send(client_encrypt_rsa(message))
    data=client_socket.recv(1024)
    temp=client_decrypt_rsa_k(data,server_public_key_part_1,server_public_key_part_2)
    print(temp)
    message="ok bob, here is a secret EPub_Bob secret "
    client_socket.send(client_encrypt_rsa(message))
    print("Deposit : 1   Withdraw : 2  Check Balance : 3  Exit : 4")
    message = input("Enter operation code (1, 2, or 3):")
    while message.lower().strip() != '4':
        if(message=="1"):
            amount = input("Enter amount u need deposit: ")
            encrypted_message = client_encrypt_rsa(message+" "+amount)
            client_socket.send(encrypted_message)
            data = client_socket.recv(1024)
            print(client_decrypt_rsa(data))
        if(message=="2"):
            amount = input("Enter amount u need withdraw: ")
            encrypted_message = client_encrypt_rsa(message+" "+amount)
            client_socket.send(encrypted_message)
            data = client_socket.recv(1024)
            print(client_decrypt_rsa(data))
        if(message=="3"):
            encrypted_message = client_encrypt_rsa(message+" 0")
            client_socket.send(encrypted_message)
            data = client_socket.recv(1024)
            print(client_decrypt_rsa(data))
        print("Deposit : 1   Withdraw : 2  Check Balance : 3  Exit : 4")
        message = input("Enter operation code (1, 2, or 3):")
```

How to handshake?

More this white diagram, after I send field verification to each other first, the CLIENT endpoint expresses skepticism to the SERVER endpoint. At this point the server end starts to authenticate itself. First behavior: send their own certification aka publish_key. at this time the client side still expresses doubt and asks for proof. So in order to prove its identity, server sends a sentence and encrypts it with the hash of this sentence. When the client receives the sentence, it first calculates the original hash of the sentence, and then decrypts it with the publish key provided by the server, and then compares the two hashes after decryption, and if they are the same, it means that the hashake is successful, and you can carry out the bank's operations.

SSL Handshake Protocol - 3

- Bob hand out his public key

A->B	Hello
B->A	Hi, I'm Bob, **Bob's-public-key**
A->B	prove it
B->A	Alice, This Is Bob E_{Priv_Bob}\{ **digest**[Alice, This Is Bob] }

Xinning Wang:

rsa:

encrypt:

client_encrypt_rsa and server_encrypt_rsa:

first transfer regular string into bits string

then apply basic rsa with key KU = [e, N], calculate C = M ^ e mod N

finally transfer the bits string into a byte string

decrypt:

client_decrypt_rsa and server_decrypt_rsa:

first transfer byte string into bits string

then apply rsa with key KR = [d, p, q], calculate M = C ^ d mod pq

finally transfer bits string into regular strings(original message)

client_decrypt_rsa_k:

this is a special version of decrypt, instead of inputing one data
and have prepared KR, like client_decrypt_rsa and server_decrypt_rsa,
it requires to enter e and N, as a way in the process of ensuring
handchecking. so from N, it would calculate two primes p and q,
and then we can use d = e^(-1) mod (p-1)(q-1) to get d,
and next calculate M = C ^ d mod n

semantical rsa:

in the base of rsa, I add a digital signature, using hash function of SHA1,
with static text "The quick brown fox jumps over the lazy dog" to xor with
the generated encrypted text and decrypted text.

part of des:

Try to do part of the des but not success, the base using simplifiec des