IMPLEMENT SSH (SECURE SHELL) SERVER

Project Report

NOVEMBER 7, 2020

SAMIR RAJESH PRAJAPATI
201851109
SANJAY RAJESH PRAJAPATI
201851086

Table of Contents

List of Figures

What is SSH?

SSH is a network protocol that allows one computer to securely connect to another computer over an unsecured network like the internet. Without encryption, data travels over the web and plaintext which makes it easy for someone to intercept username or password data and the use it. However, SSH encrypts your data through a tunnel so you can securely login to a remote machine, you can securely transmit files or safely issue remote commands and much more.

SSH is commonly implemented using the client-server model one computer is called the SSH client and another machine acts as the SSH server. SSH can then be setup using a pair of keys a public key that is stored on the SSH server and a private key that is locally stored on the SSH client. The SSH client which is usually your computer will make contact with the SSH server and provides the ID of the key pair it wants to use to prove its identity. The SSH server then creates a challenge which is encrypted by the public key and sent back to the client, you as a client then take the challenge decrypt it with your private key and send the original challenge back to the SSH server once the negotiation is complete the connection is established.

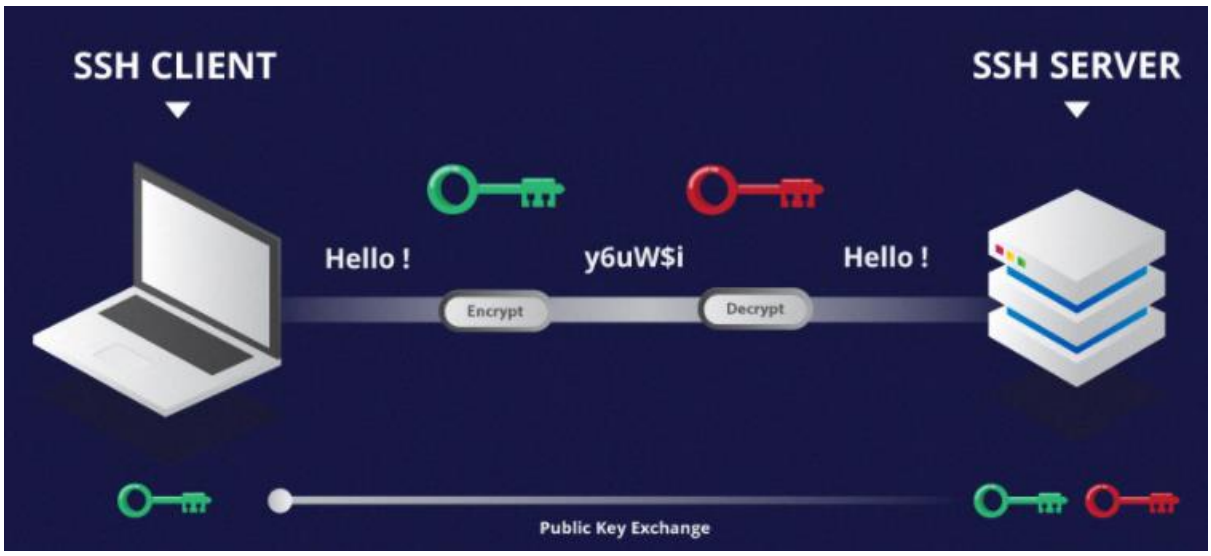

Figure 1: Communication using SSH network protocol

Brief History of SSH

"It all started 20 years ago, when I was a university researcher and there was a hacking incident in Finnish university network. Somebody installed a password sniffer on the network, a program that listens for network traffic and intercept usernames and passwords from the network traffic which hackers then used to break into other systems and when it was discovered, it had thousands of usernames and passwords in its database, including several from my company. I started thinking what could I do to make the network more secure, how could I use computers from my home, from my office, from the university without having to worry about somebody getting my password and so I started learning cryptography on networking and in the next three months wrote the program called SSH which are then published as open source in summer 95 and in a few years it basically spread throughout the whole university community, research community, enterprise, server infrastructure. Today

it's being used on hundreds of millions of systems every Linux system, every UNIX system, every Mac, browser management, network management, configuration management, file transfers, it's everywhere. More than half of world's web servers run on servers managed using SSH. The cloud infrastructure that run the services that we all use every day on the Internet is managed using SSH, it's everywhere, it's the backbone for securing the Internet and modern information systems"

- Tatu Ylonen, Inventor of SSH

RSA Cryptosystem

RSA Cryptosystem is invented by Rivest, Shamir and Adleman in 1978. It is a program which is the most frequently run programs in the world. RSA is an algorithm for asymmetric or public key encryption, as opposed to one-time pad that is symmetric and private key.

Asymmetric Encryption

Before discussion on asymmetric encryption, first we should understand why there is a need for asymmetric encryption. Lets us discuss with an example

Figure 2: Symmetric Encryption

Alice has a sensitive document that she wants to share with Bob. She uses an encryption program to protect her document with a password or passphrase that she chooses. She then sends the encrypted document to Bob. However, Bob cannot open this message because he doesn't know the passphrase that Alice used to encrypt the document. In other words: he doesn't have the key to open the lock. Now comes a real problem: how does Alice share the passphrase securely with Bob? Sending it through email is risky because others might find the passphrase and use it to decrypt any message between Alice and Bob.

Above discussed scenario is exactly the kind of problem that asymmetric encryption intends to solve. It's comparable to a mailbox on the street. The mailbox is exposed to anyone who knows its location. We can say that the location of the mailbox is completely public. Anyone who knows the address can go to the mailbox and drop in a letter. However, only the owner of the mailbox has a key to open it up and read the messages.

Now let's talk in technical details. When using asymmetric encryption, both Alice and Bob have to generate a keypair on their computers. A popular and secure way for doing this is by using the RSA algorithm. This Algorithm will generate a public and private key that are mathematically linked to each other. Public keys can be used to encrypt data and only the matching private key can be used to decrypt it. Even though the keys are linked together they cannot be derived from each other. In other words: if you know someone's public key, you cannot derive his private key. If we retake our mailbox example then the mailbox's address would be the public key something that everyone is allowed to know. The owner of the mailbox is the only one who has the private key and that is needed to open up the mailbox.

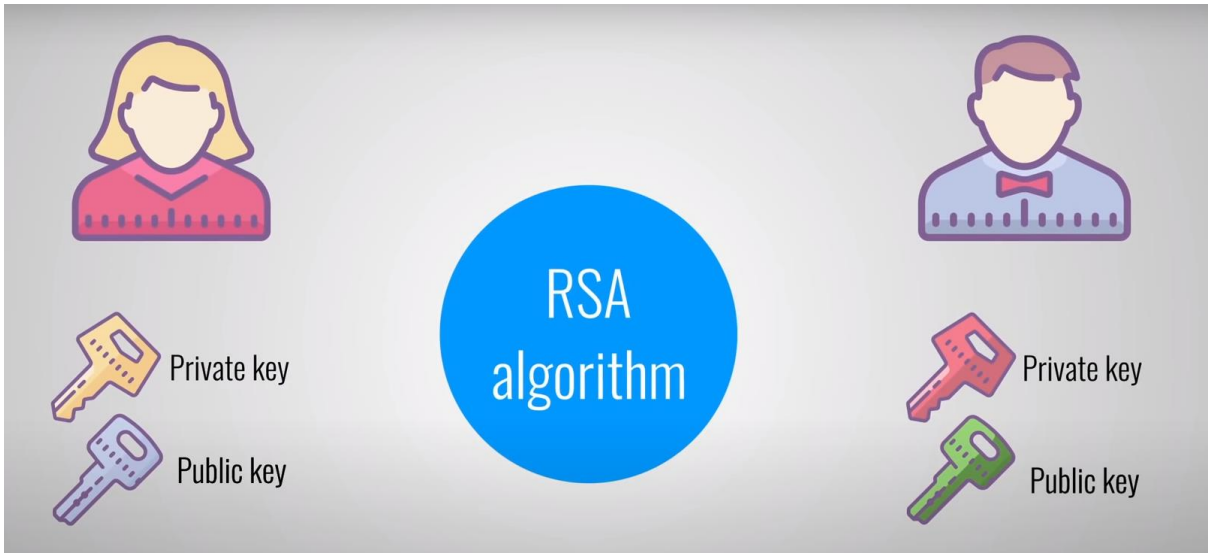

Figure 3: Asymmetric Encryption using RSA

Let's now discuss how Alice and Bob can use asymmetric encryption to communicate securely with each other. They start by exchanging their public keys. Bob gives his public key to Alice and Alice gives her public keys to Bob. Now Alice can send her sensitive document again. She takes the document and encrypts it with Bob's public key. She then sends the file to Bob, who uses his private key to unlock the document and read it. Because they use asymmetric encryption, only Bob is able to decrypt the message. Not even Alice can decrypt it because she doesn't have Bob's private key. The strength and security of the asymmetric encryption now relies on Alice and Bob to keep their private keys well protected. If an attacker steals Alice's private key, it can be used to decrypt all messages that are intended for Alice. However, the attacker cannot decrypt messages that where sent by Alice because that requires Bob's private key.

Protocol

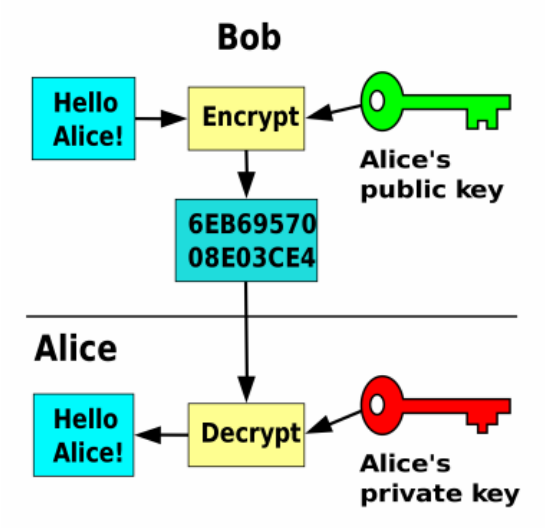

Figure 4: Protocol

- Bob generates two random keys: public key E and private key D.
- Bob publishes E for anyone to access.
- Anyone can encrypt message for Bob using E.
- Only Bob can decrypt an encrypted message using D.
- The encryption algorithm is public, so actually anyone can decrypt by trying all possible keys, but with known algorithms, it would take hundreds of years or more.

Keys

- Bob generates two big random primes p and q.
- Computes n = p*q
- Generates random e coprime with (p - 1) * (q - 1)
- Public key E is the pair (n, e)
- Private key D is the pair (p, q)

Encryption and Decryption

We take message m, which can be encoded as a sequence of bits as usual, and that is converted to an integer. And to be able to use this message, it needs to be between 0 and n-1, where n is the product of p and q which is the public key. So, we need to choose p and q big enough, so that this length in bits is sufficient for the messages we want to send. Then anyone who wants to send message m can create a cypher text c, which is equal to $c = m^e \ mod \ n$. And to do this encryption first you should use the fast-modular exponentiation algorithm.

Now to do the decryption, owner of the public key can compute or actually pre-compute a number d such that $c^d \equiv m \ mod \ n$. Now question arises how do we compute such d?

For this let's assume we already have such d and c is our ciphertext.

$$c^d \equiv m^{ed} \ mod \ n$$

So, what we need is that m to the power of ed modulo m is equal to message m itself.

$$m^{ed} \equiv m \, mod \, n$$

Now we know that n, the module, is equal to the product of two primes, p and q. And of course, those are two different primes, so they're coprime. So, we can use the Chinese remainder theorem here.

Here n = p*q, so by Chinese Remainder Theorem it is equivalent to

$$m^{ed} \equiv m \, mod \, p \, , m^{ed} \equiv m \, mod \, q$$

As p and q are prime numbers, we can use the Fermat's little theorem. And we know that with Fermat's little theorem, we can optimize the computation of modular exponent.

By Fermat's Little Theorem, $m^k \equiv m^{k \, mod \, (p-1)} \, mod \, p$, So the above condition will holds if $ed \equiv 1 \, mod \, (p-1), ed \equiv 1 \, mod \, (q-1)$ because we just instead of taking m to some exponent equal to m to the power of 1. We just take the exponent ed and say that it is going to be equal to the exponent 1 on the right-hand side of the multiplication, mod $(p-1)$ and the same for mod $(q-1)$. So, this is what we need again. This is not what is already satisfied, this is what we need. And again, to satisfy this we can make an even stronger request. We can request that $ed \equiv 1 \, mod \, (p-1)(q-1)$. If this is true then of course the above is also true. Because $(p-1)$ and $(q-1)$ are divisors of $(p-1)(q-1)$. And we know that the remainder, modular number defines the remainder module any it's divisor.

So, owner of public key can use the extended Euclid's algorithm to compute such d that $ed \equiv 1 \, mod \, (p-1)(q-1)$. Why is that? Because e is coprime with the module, so multiplication by e is invertible. And so, there is an inverse element, d, which gives 1 after multiplying by $mod \, (p-1)(q-1)$. So not only d exists, but we know the algorithm to do that and this is called the extended Euclid's algorithm.

Decryption Algorithm

- Compute $(p-1)(q-1)$.
- Compute d such that $ed \equiv 1 \, mod \, (p-1)(q-1)$ using Extended Euclid's Algorithm.
- Owner of public key can compute and store this d right after generating p, q and e.
- To decrypt ciphertext c, compute $c^d \, mod \, n$ using fast modular exponentiation

Communication Protocol

- Alice represents message m as a number between 0 and $n-1$.
- Alice computes and sends ciphertext $c = m^e \, mod \, n$
- Bob receives c and computes $m = c^d \, mod \, n$

Program of RSA Cryptosystem

Python code

```
# Install python library by running "pip install sympy"
# in command prompt
import sympy
```

```
import sys, threading

sys.setrecursionlimit(10**7)
threading.stack_size(2**27)

def ConvertToInt(message_str):
    string = 0
    for i in range(len(message_str)):
        string = string * 256 + ord(message_str[i])
    return string

def ConvertToStr(n):
    string = ""
    while n > 0:
        string += chr(n % 256)
        n //= 256
    return string[::-1]

def PowMod(a, n, mod):
    if n == 0:
        return 1 % mod
    elif n == 1:
        return a % mod
    else:
        b = PowMod(a, n // 2, mod)
        b = b * b % mod
        if n % 2 == 0:
          return b
        else:
          return b * a % mod

def ExtendedEuclid(a, b):
    if b == 0:
        return (1, 0)
    (x, y) = ExtendedEuclid(b, a % b)
    k = a // b
    return (y, x - k * y)

def InvertModulo(a, n):
    (b, x) = ExtendedEuclid(a, n)
    if b < 0:
        b = (b % n + n) % n
    return b

def Decrypt(ciphertext, p, q, exponent):
    modulo=(p-1)*(q-1)
    d=InvertModulo(exponent,modulo)
    return ConvertToStr(PowMod(ciphertext, d, p * q))
```

```
def Encrypt(message, modulo, exponent):
    return PowMod(ConvertToInt(message), exponent, modulo)

p = sympy.randprime(10**500, 10**700)
q = sympy.randprime(10**500, 10**700)
exponent = sympy.randprime(10**600, 10**700)
modulo = p * q
message="A computer network is a group of computers that use a set of comm
on communication protocols over digital interconnections for the purpose o
f sharing resources located on or provided by the network nodes."

ciphertext = Encrypt(message, modulo, exponent)
print(f"Ciphertext:- {ciphertext}")
decrypt_message = Decrypt(ciphertext, p, q, exponent)
print(f"Decrypt Message:- {decrypt_message}")
```

Output

$ python RSA.py

Ciphertext:-
89077705903236373040931233934000073061510194780286611446223641178109339976
02819886048273134934323713281575816358358877139049701048735141143403423843
65300615948111863419739750784456748341365577386279985631829352918639722105
58085685928760580398946236887662891006660794699732908370147410979947240110
17015315046412876084581707053058992407492210896365944787926076118946159724
27387884366906093497709255412763829177340217085573540227713527893757096651
34367798802461853172109352745503561699453186116648093546864118897720905701
35604469210545274777002414951209721852189950205759262120451458274760668932
82355884862866038125552819242347218508605728074193798742437622208792227313
23921250035211623668195739112674462061164717650512306817489242601050606343
45103093474864710708119479315677928579783460596824528496947840902004292749
07085273618361579276213618141900251679894808898834077481327707547297778452
30081166100593058502902662041607255354714955482952595573987601932711258774
19271719617634927224327596085421926989572042454920428232803703409661600137
33355530803197524958658756226680827773821627852424710046514461115815641236
18999477422474230039326237028194569409387377951836305000813325327840146518
21303959527493568502948766674345100789226394982892917120545841431979926862
66475353902632475723555929219402675880959289925287101815709813535172457534
42516498766363471830558200233388103338352530074785487256703847713 72

Decrypt Message: - A computer network is a group of computers that use a set of common communication protocols over digital interconnections for the purpose of sharing resources located on or provided by the network nodes.

Diffie–Hellman key exchange

Diffie–Hellman key exchange is a method of securely exchanging cryptographic keys over a public channel. Named after Whitfield Diffie and Martin Hellman.

Traditionally, secure encrypted communication between two parties required that they first exchange keys by some secure physical means, such as paper key lists transported by a trusted courier. The Diffie–Hellman key exchange method allows two parties that have no prior knowledge of each other to jointly establish a shared secret key over an insecure channel. This key can then be used to encrypt subsequent communications using a symmetric key cipher.

Key Sharing Algorithm

- Server and Client get public numbers p and g where p is a prime number and g is a generator of p.
- Client selected a private key a and Server selected a private key b.
- Client and Server compute public values
  - Client: A = $g^a \ mod \ p$
  - Server: B = $g^b \ mod \ p$
- Client and Server exchange public values.
- Client receives public value B and Server receives public value A.
- Client and Server compute symmetric keys
  - Client: $B^a \ mod \ p$
  - Server: $A^b \ mod \ p$
- $B^a \ mod \ p \ \equiv A^b \ mod \ p$ is the shared secret.

Example

Step 1: Client and Server get public numbers p = 31, g = 17

Step 2: Client selected a private key a = 7 and

Server selected a private key b = 4

Step 3: Client and Server compute public values

Client: A = $g^a \ mod \ p = 17^7 \ mod \ 31 = 12$

Server: B = $g^b \ mod \ p = 17^4 \ mod \ 31 = 7$

Step 4: Client and Server exchange public values

Step 5: Client receives public value B = 7 and

Server receives public value A = 12

Step 6: Client and Server compute symmetric keys

Client: symmetric key = $B^a \ mod \ p = 7^7 \ mod \ 31 = 28$

Server: symmetric key = $A^b \ mod \ p = 12^4 \ mod \ 31 = 28$

Step 7: 28 is the shared secret.

How SSH Works?

SSH protocol uses symmetric encryption, asymmetric encryption and hashing in order to secure transmission of information. The SSH connection between the client and the server happens in three stages:

1. Verification of the client by the server.
2. Generation of a session key to encrypt all the communication.
3. Authentication of the client.

Now, we will discuss about these stages.

Verification of Client

The client initiates a SSH connection with the server. Server listens to default port 22 (this port can be changed) for SSH connections. At this point, the server identity is verified. There are two cases:

1. The KNOWN_HOSTS file does not contain the information about the client which is accessing the Server.
    a. Server will show error message “User not Found”.
2. The KNOWN_HOSTS file contain the information about the client which is accessing the Server.
    a. Server will show message “User Found”.

Generation of Session Key

After the client is verified, both the parties negotiate a session key using a version of something called the Diffie-Hellman algorithm (explained above). This algorithm is designed in such a way that both the parties contribute equally in generation of session key. The generated session key is shared symmetric key which will be used in symmetric encryption i.e. the same key is used for encryption and decryption.

Authentication of the Client

The final stage involves authentication of the client. Authentication is done using SSH key pair. As the name suggests, SSH key pair (explained above) is nothing but a pair of two key to serve two different purposes. One is public key that is used to encrypt data and can be freely shared. The other one is private key that is used to decrypt data and is never shared with anyone.

After symmetric encryption has been established, the authentication of the client happens as follows:

1. The client begins by sending an ID (we have used username) for the key pair it would like to authenticate with to the server.
2. If a public key with matching ID is found in the KNOWN_HOSTS file, the server generates a random number and uses the public key to encrypt the number and sends this encrypted message.

3. If the client has the correct private key, it will decrypt the message to obtain the random number that was generated by the server.
4. The client encrypts the obtained random number with the key created from hash of shared session key.
5. The client then sends this encrypted data to the server as an answer to the encrypted number message.
6. The server uses the same key, created from hash of shared session key, to decrypt the received data from the Client. It compares decrypt number with its own previously generated random number. If these two numbers are matched, it proves that the client was in possession of the private key and the client is authenticated.

Flowchart

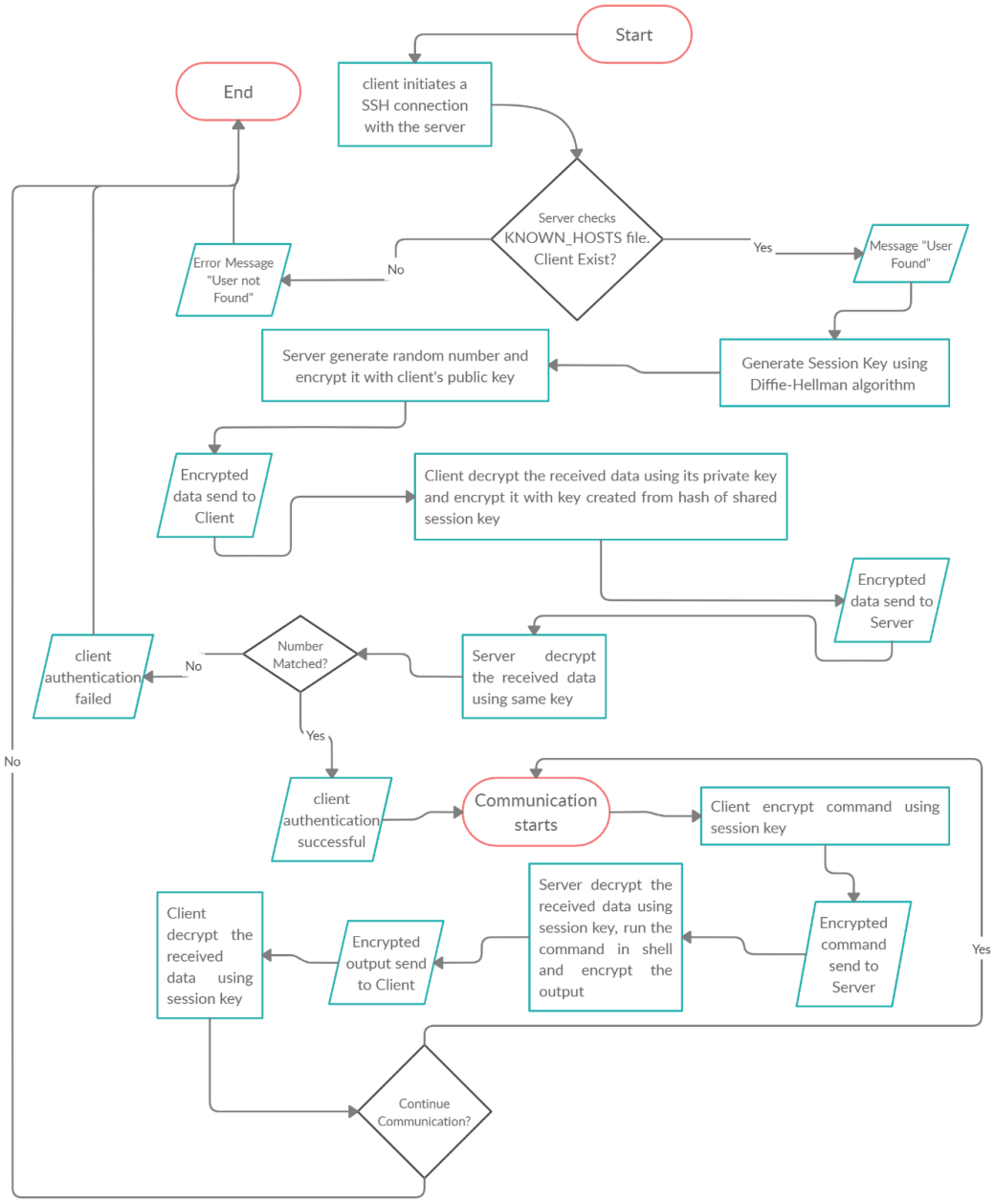

Figure 5: Flowchart

Structure of our project

Project_Folder

|_ client

|_Client.java

|_KEY.txt

|_RSA.java

|_SymmetricCrypto.java

|_ server

|_Server.java

|_KNOWN_HOSTS.txt

|_RSA.java

|_SymmetricCrypto.java

Steps to run project

Step 1

First go to server folder.

Open terminal in that folder.

Compile Server.java file by running following command

```
javac Server.java
```

Run Server.java file with the following format.

```
java Server
```

Step 2

Now go to client folder.

Open terminal in that folder.

Compile Client.java file by running following command

```
javac Client.java
```

Run Client.java file with the following format.

java Client username@ipaddress

Example:

```
java Client sanjay@127.0.0.1
```

If connection establish successfully and user is verified then you are logged in to the server.

Now you can access the server from your terminal.

Project Code

Client.java

```
// Importing Required Classes
import java.net.Socket;
import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.math.BigInteger;
import java.nio.file.Paths;
import java.security.SecureRandom;
import java.util.List;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;

public class Client {

    private Socket socket;
    private DataInputStream dataInputStream;
    private DataOutputStream dataOutputStream;
    private BufferedReader bufferedReader;

    private int bitLength = 1024;
    private BigInteger USER_PublicKey_e;
    private BigInteger USER_PublicKey_N;
    private BigInteger USER_PublicKey_d;
    private RSA rsa = new RSA();
    private SymmetricCrypto sym = new SymmetricCrypto();

    Client(String IP, String USERNAME) throws Exception {

        try {
            socket = new Socket(IP, 22);
        } catch (Exception e) {
```

```
            System.out.println(e);
            System.out.println("Connection Failed!!! Check the IP Address
of the server...");
            return;
        }
        System.out.println("\nConnection Established\nVerifying User...");

        dataInputStream = new DataInputStream(socket.getInputStream());
        dataOutputStream = new DataOutputStream(socket.getOutputStream());
        bufferedReader = new BufferedReader(new InputStreamReader(System.i
n));

        // Step 1      - username
        dataOutputStream.writeUTF(USERNAME);
        dataOutputStream.flush();

        String ACK = dataInputStream.readUTF();
        if(ACK.equals("not found")) {
            System.out.println("Username not Found!");
            return;
        } else {
            Path path = Paths.get("KEY.txt");
            List<String> lines = Files.readAllLines(path, StandardCharsets
.UTF_8);

            USER_PublicKey_e = new BigInteger(lines.get(0).split("=")[1]);
            USER_PublicKey_N = new BigInteger(lines.get(1).split("=")[1]);
            USER_PublicKey_d = new BigInteger(lines.get(2).split("=")[1]);
        }

        // Step 2       - p and g
        BigInteger p = new BigInteger(dataInputStream.readUTF());

        BigInteger g = new BigInteger(dataInputStream.readUTF());

        // Step 3       - a
        SecureRandom secureRandom = new SecureRandom();
        BigInteger a = BigInteger.probablePrime(bitLength, secureRandom);

        // Step 4       - A and B       /* A = g^a mod p */
        BigInteger A = g.modPow(a, p);
        BigInteger B = new BigInteger(dataInputStream.readUTF());

        dataOutputStream.writeUTF(A.toString());
        dataOutputStream.flush();

        // Step 5       - S            /* s = B^a mod p */
        String SecretKey = B.modPow(a, p).toString();

        // Step 6               - Encrypted n received and decrpt and e

        String len = dataInputStream.readUTF();

        byte[] encryptedN = new byte[Integer.parseInt(len)];
```

```
        dataInputStream.read(encryptedN, 0, Integer.parseInt(len));

        byte[] decryptedN = rsa.decryptMessage(encryptedN, USER_PublicKey_
d, USER_PublicKey_N);

        dataOutputStream.writeUTF(sym.encrypt(new String(decryptedN), Secr
etKey));
        dataOutputStream.flush();

        ACK = dataInputStream.readUTF();
        if(ACK.equals("not verified")) {
            System.out.println("Permission Denied!");
            return;
        } else {
            System.out.println("User Verified");
        }

        System.out.println("You are login to the Server...");
        System.out.println("Type \"exit\" to logout and exit the Server");
        System.out.print(USERNAME + "@ ");


        bufferedReader = new BufferedReader(new InputStreamReader(System.i
n));
        String command = bufferedReader.readLine();
        while(!command.equals("exit")) {

            command = sym.encrypt(command, SecretKey);
            dataOutputStream.writeUTF(command);
            dataOutputStream.flush();

            String result = sym.decrypt(dataInputStream.readUTF(), SecretK
ey);
            System.out.print(result);
            System.out.print(USERNAME + "@ ");

            command = bufferedReader.readLine();
        }
        command = sym.encrypt(command, SecretKey);
        dataOutputStream.writeUTF(command);
        dataOutputStream.flush();

        socket.close();
        dataInputStream.close();
        dataOutputStream.close();
        System.out.println("Connection Closed.");
    }

    public static void main(String[] args) throws Exception {
        String username, ip;
        if(args.length == 0) {
            System.out.println("Please enter the username and ip of the se
rver...");
            System.out.println("Like this: username@ipaddress");
```

```
            return;
        } else {
            String input[] = args[0].split("@");
            if(input.length != 2) {
                System.out.println("Invalid username and ip address format
!");
                System.out.println("Like this: username@ipaddress");
                return;
            }
            else {
                username = input[0];
                ip = input[1];
            }
        }
        Client client = new Client(ip, username);
    }
}
```

Server.java

```
// Importing Required Classes
import java.net.Socket;
import java.net.URI;
import java.net.ServerSocket;
import java.io.IOException;
import java.io.InputStreamReader;
import java.math.BigInteger;
import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.nio.file.Paths;
import java.security.SecureRandom;
import java.util.List;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;

public class Server {

    private ServerSocket serverSocket;
    private Socket socket;
    private DataInputStream dataInputStream;
    private DataOutputStream dataOutputStream;
    private BufferedReader bufferedReader;

    private int bitLength = 1024;
    private String USERNAME;
    private BigInteger USER_PublicKey_e;
    private BigInteger USER_PublicKey_N;
    private RSA rsa = new RSA();
    private SymmetricCrypto sym = new SymmetricCrypto();
    private Process process;
```

```
    Server(int PORT) throws Exception {

        // Starting Server Socket at Port PORT
        serverSocket = new ServerSocket(PORT);
        System.out.println("\nServer ON.");
        System.out.println("Waiting for Client request...");
        // Waiting for Client
        socket = serverSocket.accept();
        System.out.println("Client Request Received.");
        System.out.println("\nConnection Established\n");

        dataInputStream = new DataInputStream(socket.getInputStream());
        dataOutputStream = new DataOutputStream(socket.getOutputStream());

        // Step 1           - username and check
        USERNAME = dataInputStream.readUTF();
        System.out.println("username received: " + USERNAME);

        Path path = Paths.get("KNOWN_HOSTS.txt");
        List<String> lines = Files.readAllLines(path, StandardCharsets.UTF
_8);
        boolean isFound = false;
        for(int i=0; i<lines.size(); i=i+1) {
            if(lines.get(i).startsWith(USERNAME + ":")) {
                isFound = true;
                String line[] = lines.get(i).split(":");
                USER_PublicKey_e = new BigInteger(line[1].split("=")[1]);
                USER_PublicKey_N = new BigInteger(line[2].split("=")[1]);
            }
        }

        if(!isFound) {
            System.out.println("Username not Found!");

            dataOutputStream.writeUTF("not found");
            dataOutputStream.flush();
            System.out.println("ACK sent");

            return;
        } else {
            System.out.println("Username Found!");
            // System.out.println(USER_PublicKey_e);
            // System.out.println(USER_PublicKey_N);

            dataOutputStream.writeUTF("found");
            dataOutputStream.flush();
            System.out.println("ACK sent");
        }

        // Step 2           - p and g
        SecureRandom secureRandom = new SecureRandom();
        BigInteger p = BigInteger.probablePrime(bitLength, secureRandom);
        BigInteger g = BigInteger.probablePrime(bitLength, secureRandom);
```

```
        dataOutputStream.writeUTF(p.toString());
        dataOutputStream.flush();
        System.out.println("p sent");

        dataOutputStream.writeUTF(g.toString());
        dataOutputStream.flush();
        System.out.println("g sent");

        // Step 3            - b
        BigInteger b = new BigInteger(bitLength, secureRandom);

        // Step 4            - A and B    /* B = g^b mod p */
        BigInteger B = g.modPow(b, p);
        dataOutputStream.writeUTF(B.toString());
        dataOutputStream.flush();
        System.out.println("B sent");

        BigInteger A = new BigInteger(dataInputStream.readUTF());
        System.out.println("A received");

        // Step 5              - S        /* s = A^b mod p */
        String SecretKey = A.modPow(b, p).toString();
        System.out.println("SecretKey is: " + SecretKey);

        // Step 6               - Encrypt n

        String n = new BigInteger(bitLength/10, secureRandom).toString();
        byte byt[] = n.getBytes();
        byte[] encryptedN = rsa.encryptMessage(byt, USER_PublicKey_e, USER
_PublicKey_N);

        dataOutputStream.writeUTF(Integer.toString(encryptedN.length));
        dataOutputStream.flush();
        System.out.println("len sent");

        dataOutputStream.write(encryptedN, 0, encryptedN.length);
        dataOutputStream.flush();
        System.out.println("Encrypted n array sent: " + n);

        String encrytedNwithS = dataInputStream.readUTF();
        String decryptedNwithS = sym.decrypt(encrytedNwithS, SecretKey);
        System.out.println("decryptedNwithS received: " + decryptedNwithS)
;

        if(decryptedNwithS.equals(n)) {
            dataOutputStream.writeUTF("verified");
            dataOutputStream.flush();
            System.out.println("verified");
        } else {
            dataOutputStream.writeUTF("not verified");
            dataOutputStream.flush();
            System.out.println("not verified");
        }
```

```
        String command = dataInputStream.readUTF();
        command = sym.decrypt(command, SecretKey);

        StringBuilder result;
        while(!command.equals("exit")) {

            try {
                System.out.println(command + " command ran");
                process = Runtime.getRuntime().exec(command);
            } catch (Exception e) {
                dataOutputStream.writeUTF(sym.encrypt("Command not found!\
n", SecretKey));
                dataOutputStream.flush();

                command = dataInputStream.readUTF();
                command = sym.decrypt(command, SecretKey);
                continue;
            }
            bufferedReader = new BufferedReader(new InputStreamReader(proc
ess.getInputStream()));
            result = new StringBuilder();
            String output;

            output = bufferedReader.readLine();
            while(output != null) {
                result.append(output + "\n");
                output = bufferedReader.readLine();
            }

            dataOutputStream.writeUTF(sym.encrypt(result.toString(), Secre
tKey));
            dataOutputStream.flush();
            process.waitFor();
            process.exitValue();
            process.destroy();

            command = dataInputStream.readUTF();
            command = sym.decrypt(command, SecretKey);
        }

        socket.close();
        dataInputStream.close();
        dataOutputStream.close();
        System.out.println("Connection Closed.");
    }

    public static void main(String[] args) throws Exception{
        int PORT = 22;
        Server server = new Server(PORT);
    }
}
```

RSA.java

```
import java.math.BigInteger;

public class RSA
{

    // Encrypting the message
    public byte[] encryptMessage(byte[] message, BigInteger e, BigInteger
N)
    {
        return (new BigInteger(message)).modPow(e, N).toByteArray();
    }

    // Decrypting the message
    public byte[] decryptMessage(byte[] message, BigInteger d, BigInteger
N)
    {
        return (new BigInteger(message)).modPow(d, N).toByteArray();
    }

}
```

SymmetricCrypto.java

```
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Base64;

import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;

class SymmetricCrypto {

    public String encrypt(String data, String SecretKey) throws Exception{
        byte[] SecretHashedKey = createMD5(SecretKey);
        SecretKey secKey = new SecretKeySpec(SecretHashedKey, "AES");
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.ENCRYPT_MODE, secKey);

        byte[] newData = cipher.doFinal(data.getBytes());
        byte[] encryptedData = Base64.getEncoder().encode(newData);

        return new String(encryptedData);
    }

    public String decrypt(String encryptedData, String SecretKey) throws E
xception {
        byte[] SecretHashedKey = createMD5(SecretKey);
        SecretKey secKey = new SecretKeySpec(SecretHashedKey, "AES");
```

```
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.DECRYPT_MODE, secKey);

        byte[] decryptedData=cipher.doFinal(Base64.getDecoder().decode(enc
ryptedData.getBytes()));

        return new String(decryptedData);
    }

    public byte[] createMD5(String key) throws NoSuchAlgorithmException {
        MessageDigest md = MessageDigest.getInstance("MD5");
        md.update(key.getBytes());
        byte byteData[] = md.digest();
        return byteData;
    }
}
```

KEY.txt

```
e=92824765019659283237338966066041216643490783122149485559305531113204028318394806352831856313892770206789559186093637612635468132730766807369537346741870
47

N=10783812774448330685727963771972799100615888444754906702746088691816701593344181686853117400341153377029809014498365822715823579655297782269528836418088958399329728595134685723215448371664965618273656755202470815411510437954224897626772742400508938404328181171521358661922281528616976569150962097215511064387187626598905963675313737283359955772234855413478848654068065568594444957316074281580959619073804122968916188841967371349433435371211653177054556111065945036326038058820588041572757328143522767136974148781841204999654356624471370785265273969592837905553055753814631939544800892822877720844185186170480786995159

d=85994163159053154606141683547717199513199086309521036681590621114417539316651258045673883262278708545937839648428275879805812266157439347036512617355754597557445336721342684239861177406936813253240871234737170765911498301218251453077450390167197133193706205763179669971743671098532102984012556001317712614028059339923178117496432612350848188679866198444898747367189693936528538835914718031244577316399842337631920776709140688024407010108797730721712233732268299351601333145462050290512969148319931020573214518576716791357111832913184221220704862660294101728358830901522322971750658629750390146390539855311058804687
```

KNOWN_HOSTS.txt

```
USERNAME:PUBLIC_KEY

sanjay:e=92824765019659283237338966066041216643490783122149485559305531113204028318394806352831856313892770206789559186093637612635468132730766807369537346741870467:N=10783812774448330685727963771972799100615888444754906702746088691816701593344181686853117400341153377029809014498365822715823579655297782269528836418088958399
```

93297285951346857232154483716649656182736567552024708154115104379542248976267727
42400508938404328181171521358661922281528616976569150962097215511064387187626598
90596367531373728335995577223485541347884865406806556859444495731607428158095961
90738041229689161888419673713494334353712116531770545561110659450363260380588205
88041572757328143522767136974148781841204999654356624471370785265273969592837905
55305575381463193954480089282287772084418518617048078699515 9

Screenshots

From Server.java

```
praja@LAPTOP-DPO5RVO1 MINGW64 /d/Sanjay/IIITV/Sem 5/Computer Networks/SHH Project/Final 1.0/server
$ javac Server.java

praja@LAPTOP-DPO5RVO1 MINGW64 /d/Sanjay/IIITV/Sem 5/Computer Networks/SHH Project/Final 1.0/server
$ java Server

Server ON.
Waiting for Client request...
Client Request Received.

Connection Established

username received: sanjay
Username Found!
ACK sent
p sent
g sent
B sent
A received
SecretKey is: 10228550499547877644203189816519648311873325183438309851701270167963283271552424898281448530616434131131789636607829916493038663997386789631420842788604472439389515268436900312843037965870810486073697069043187266504720519244350887766348891493356740008844550164409378912535767066382283347885859583420800141577 8
len sent
Encrypted n array sent: 157531485154003562389112957 43
decryptedNwithS received: 157531485154003562389112957 43
verified
ls command ran
Connection Closed.
```

Figure 6: Output from Server

From Client.java

```
praja@LAPTOP-DPO5RVO1 MINGW64 /d/Sanjay/IIITV/Sem 5/Computer Networks/SHH Project/Final 1.0/client
$ javac Client.java

praja@LAPTOP-DPO5RVO1 MINGW64 /d/Sanjay/IIITV/Sem 5/Computer Networks/SHH Project/Final 1.0/client
$ java Client sanjay@127.0.0.1

Connection Established
Verifying User...
User Verified
You are login to the Server...
Type "exit" to logout and exit the Server
sanjay@ ls
KNOWN_HOSTS.txt
RSA.class
RSA.java
Server.class
Server.java
SymmetricCrypto.class
SymmetricCrypto.java
sanjay@ exit
Connection Closed.
```

Figure 7: Output from Client